AD-A276 629

PART 2 OF 2

Office of Naval Research
Grant N00014-89-J-1795

DTIC
ELECTE
MAR 09 1994
S E D

Final Report

Noninvasive Spectral Analysis and Beam Mapping of Focused Ultrasound

by
Walter Mayer (Principal Investigator)
Thomas Neighbors
Herbert Ruf
John Wolf
Qicao Zhu

Ultrasonics Research Laboratory
Physics Department
Georgetown University
Washington, DC 20057

January 1994

Reproduction in whole or in parts permitted for any purpose
of the United States Government

Approved for public release

94-05139

94 2 16 008

Summary Page IV

Appendices

This appendix section contains the various codes that were used to do the numerical analysis discussed in Section I.

Also included here are two papers which were prepared to conclude the work carried out during an earlier period of the Grant, preceding the work on focused beams. This work was to be in preparation for an anticipated later effort to merge the focused beam work and the earlier work on plate excitation by bounded, but non-focused beams. Unfortunately, the results presented here could not be used in conjunction with the results presented in Section I due to the decision not to continue the Grant.

Accesion For		
NTIS CRA&I ☒ DTIC TAB ☐ Unannounced ☐ Justification per A276211		
By Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

APPENDIX A

LFOCUS.EXE

USER'S GUIDE

A.1. INTRODUCTION

This appendix outlines the steps necessary for using the line focus software, **LFOCUS.EXE**, developed to support ONR research on optical probing of ultrasonic fields. Section A.2 outlines the hardware requirements for program execution. Since the software is menu driven rather than batch processed, Section A.3 provides a short user's guide for the menu driven interface. Section A.4 describes the output file structure used for data post processing.

A.2. Hardware Requirements

The minimum hardware configuration required to operate **LFOCUS.EXE** consists of a computer with an Intel 80386 Central Processing Unit (CPU) or 80486 CPU with the following features:

- one high density, 3½" or 5¼" floppy disk drive,
- PC Disk Operating System (DOS) Version 3.3 or later[1],
- Monochrome display or display adapter,
- Intel 80387 Math CO-Processor[2],
- A hard drive with 40 to 50 Mega-Bytes (Mb) of available storage, and

---

1 Program compatibility has been verified with MS-DOS v 3.3 and 4.01 and Digital Research DOS v 5.0 and 6.0.

2 The 80486 chip has an integral co-processor.

- 4.0 Mb of extended memory.

LFOCUS.EXE accesses the Intel co-processor and will not run without it. As an alternative, the program can be compiled to access the Weitek, ABACUS chip. This change could improve program execution speed by about a factor of two. A graphics monitor does not improve program performance. Additional hard disk storage space is only useful for multiple program runs. A Microsoft® or compatible mouse can be used as a pointing device for program execution; however, a mouse is not required for program execution.

A.3. Initial Execution

To run LFOCUS.EXE for the first time change to the hard disk directory which contains the program, type LFOCUS, and press <Enter>. LFOCUS.EXE loads and presents the initial menu screen shown in Figure A.1.

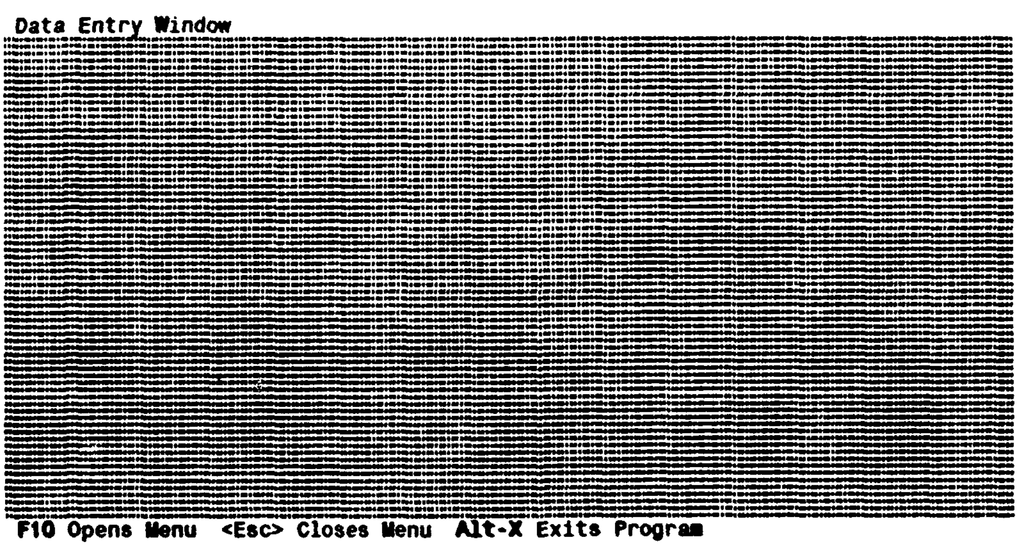

Figure A.1. LFOCUS.EXE Initial Menu

If a mouse is present, move the mouse cursor over the desired function and click the left mouse button to execute the function. If a mouse is not present, press the highlighted key to execute the function. For example, the initial menu can be activated by pressing the Alt key and W to select the Data Entry Window, moving the mouse cursor over Data Entry Window and clicking the left mouse button, or pressing **F10**. If the Data Entry Window function is executed by either technique, the first pull-down menu appears as shown in Figure A.2.

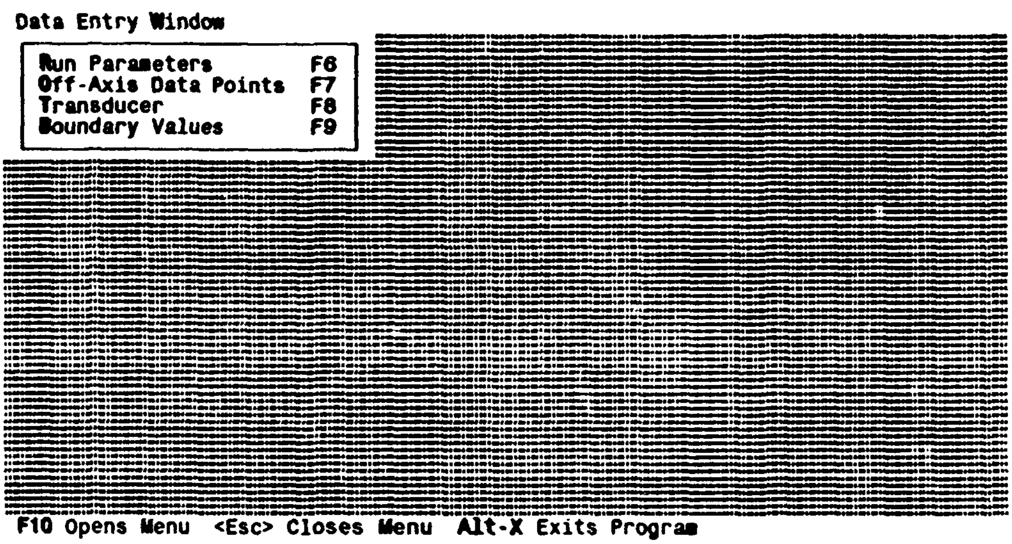

Figure A.2. Data Entry Options

Run Parameters (F6) specifies the number of spectra used in the calculations (minimum and maximum) and the on and off-axis data output points. Off-Axis Data Points (F7) specifies the on-axis locations where the off-axis data points are output to a disk file. Transducer (F8) specifies the key parameters associated with the line focus transducer, such as, dimensions, aperture angle, initial on-axis pressure, etc.. Boundary Values (F9) provides options for specifying the pressure amplitude distribution at the transducer surface. To activate a menu press the highlighted key, the specified function key, or use the mouse cursor and click on the choice.

A.3.1. Run Parameters

To specify the input parameters for a run press **R**, function key **F6**, or highlight the **Run Parameters** option with the mouse cursor and click. The **Run Parameters** menu shown in Figure A.3 appears. The parameter values displayed are those used the last time the program was executed. To exit this menu without changing any values, use the mouse cursor to click the [■] symbol or press **<Enter>**. To move through the parameters on the menu without a mouse, press the **Tab** key. Each time the **Tab** key is pressed the next menu option is highlighted for data entry. To back-up, press **Shift Tab** and the previous data entry option is highlighted for data entry. To select a field for data entry move the mouse cursor over the text describing the value and click.

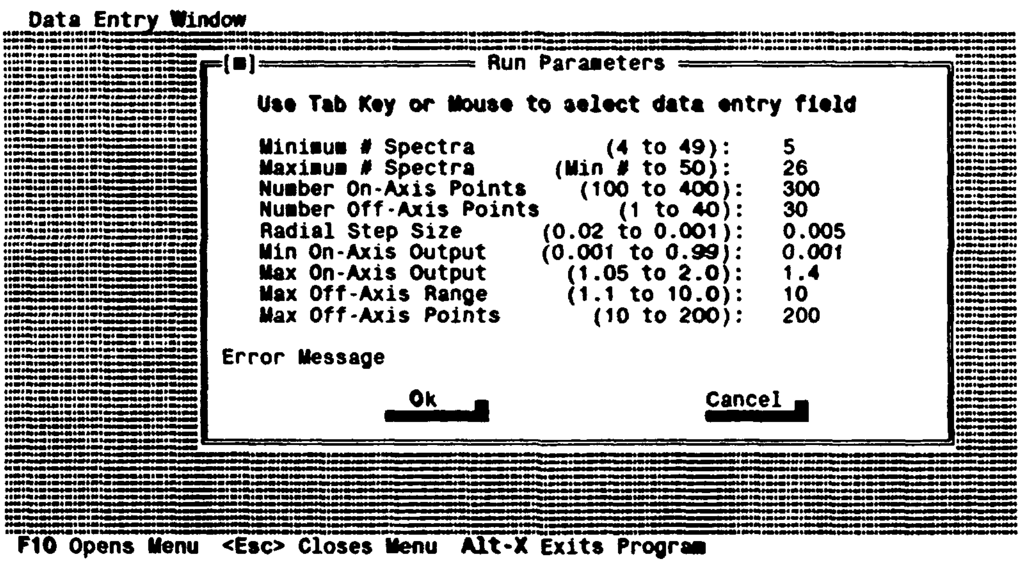

Figure A.3. Run Parameters Menu

To enter in a new value key in the appropriate number. **Do not press Enter, if you wish to change other values before exiting the menu.** Pressing **<Enter>** will exit the menu, if there are no errors. If a value is keyed in which is out of range, an error message appears when you attempt to exit the menu. For example, **Minimum # Spectra** represents the number of spectral terms present at the start of a calculation. As shown, this is allowed to

take on a value between 4 and 49 terms. If a value outside of this range is keyed in, such as 2, the program writes the error message shown in Figure A.4 when the <Enter> key is pressed. The program will not exit the menu until the entry is corrected. If multiple errors are made when keying in values, each error has to be corrected.

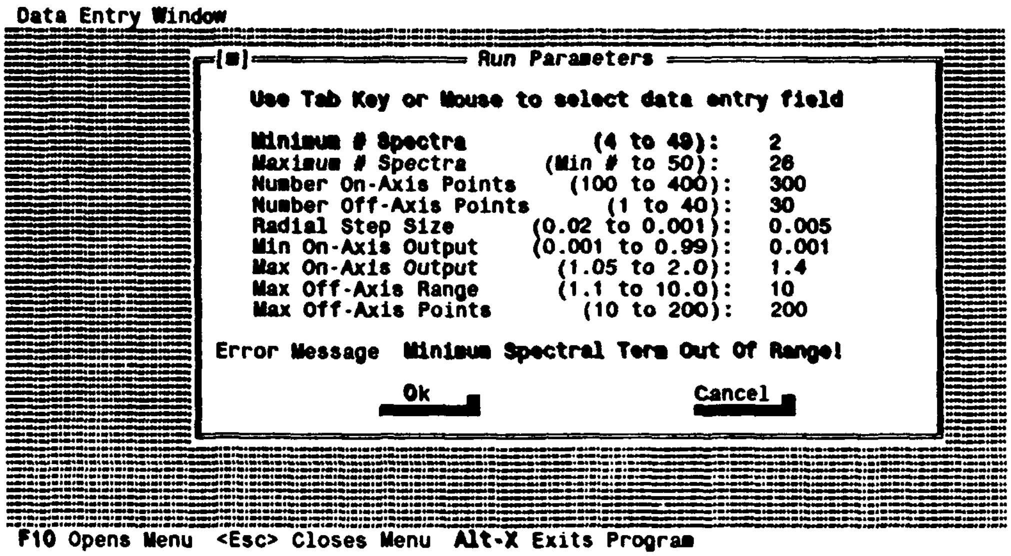

Figure A.4. Typical Error Message

The data entry values covered in the **Run Parameters** menu include:

Minimum # Spectra	minimum number of spectral terms present at the start of a calculation,
Maximum # Spectra	maximum number of spectral terms allowed in the calculation, that is the number of terms allowed before the Fourier spectra is truncated,
Number On-Axis Points	number of on-axis points where the amplitude and phase spectra are saved during the calculation,
Number Off-Axis Points	number of points on the axis where the off-axis amplitude and phase spectra are saved during the calculation,
Radial Step Size	normalized, transverse mesh step size used in the calculation,

Min On-Axis Output	the on-axis point where the on-axis output starts (0 is at the face of the transducer and 1 is at the geometric focal point),
Max On-Axis Output	the on-axis point where the calculation is terminated and the last on-axis data values are written to a file,
Max Off-Axis Range	maximum normalized size the transverse grid is allowed to expand to during the calculations, and
Max Off-Axis Points	maximum number of off-axis points which are output at each specified output location.

The values shown in Figures A.3 and A.4 within the brackets are the allowable data entry ranges.

A.3.2. Off-Axis Data Point Menus

To review or change the on-axis locations selected for off-axis data output prior to running LFOCUS.EXE, select the data entry window and press O, F7, or use the mouse cursor to select **Off-Axis Data Points** from the options shown in Figure A.2. At this point, the **Off Axis Location 1** menu shown in Figure A.5 appears. Use the **Tab** or **Shift Tab** keys to move between the data entry fields. To exit without changing any values use the mouse to click on the [■] symbol or the **Ok** or **Cancel** buttons. Alternatively, use the **Tab** or **Shift Tab** keys to move to the **Ok** or **Cancel** buttons. The range of allowable off-axis data values is determined by the **Run Parameters** menu. For example, in menu shown in Figure A.3, the maximum on-axis output point is 1.4 (40% beyond the geometric focal point for the transducer). Data entry follows the technique previously described for the **Run Parameters** menu. Remember, the software will not exit a menu which has a value out of range. Also, the program expects the values to be monotonically increasing. This menu will continue to appear until all points have been selected for off-axis data output. Once the last data point has been selected, the program returns to the initial data entry menu shown in Figure A.1.

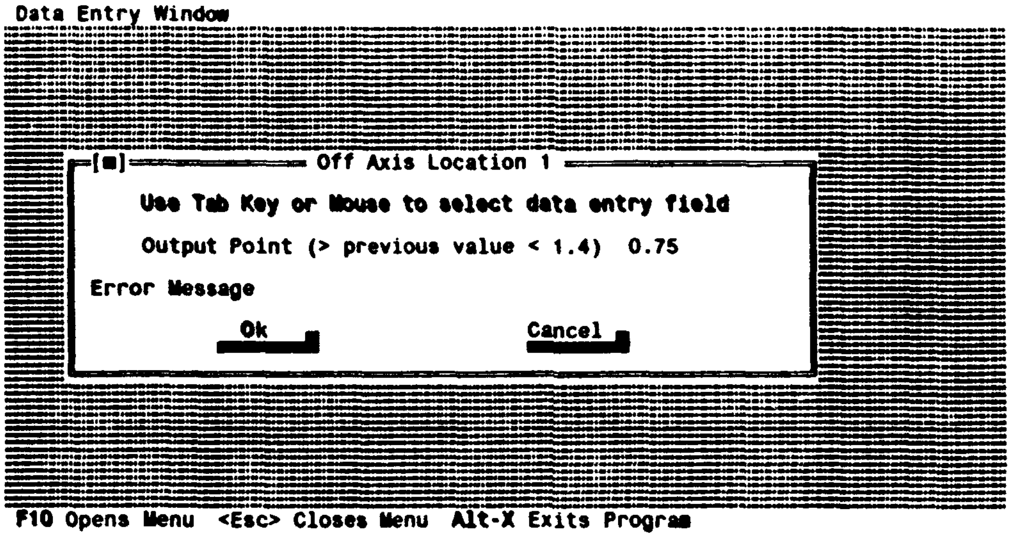

Figure A.5. Off-Axis Data Point Selection

A.3.3. Transducer Characteristics Menu Data Entry

To review or change the transducer data values prior to running **LFOCUS.EXE** select the data entry window and press **T**, **F8**, or use the mouse cursor to select **Transducer** from the options shown in Figure A.2. At this point the **Transducer Characteristics** menu shown in Figure A.6 appears. Use the **Tab** or **Shift Tab** keys to move between the data entry fields. To exit without changing any values use the mouse to click on the [■] symbol or the **Ok** or **Cancel** buttons. Alternatively use the **Tab** or **Shift Tab** keys to move to the **Ok** or **Cancel** buttons. Data entry follows the technique previously described for the **Run Parameters** menu. Remember, the software will not allow the user to exit a menu which has a value out of range. Once the last data entry has been made the program returns to the initial data entry menu shown in Figure A.1.

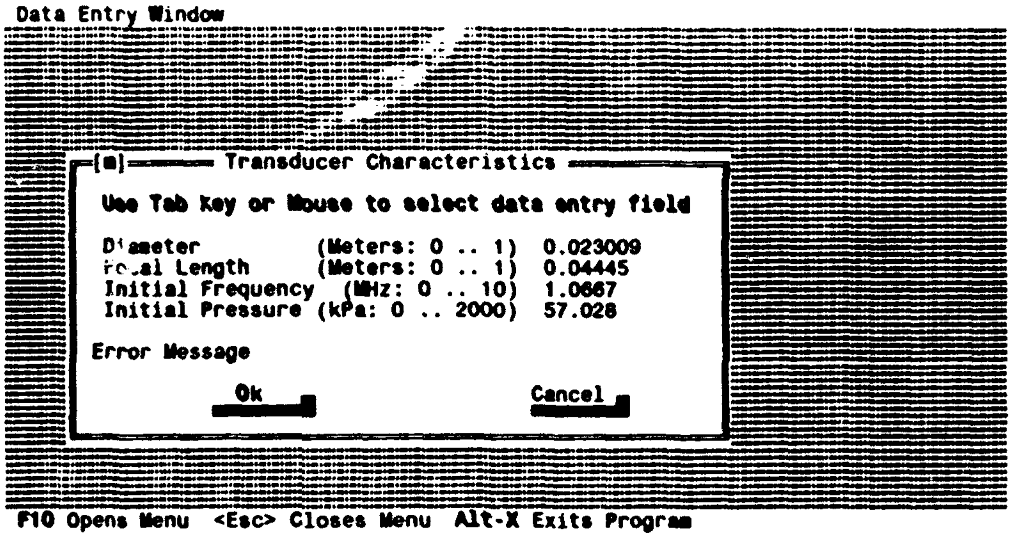

Figure A.6. Transducer Data Entry

A.3.4. Boundary Value Menus

A three level menu system is used for selecting an existing or entering a new normalized pressure distribution across the face of the line focus transducer. To select this option from the choices presented in Figure A.2, press **B**, **F9**, or highlight **Boundary Value** with the mouse cursor and click. The **Transducer Pressure Distribution Options Menu** shown in Figure A.7 appears. This menu provides three choices for the pressure distribution at the transducer surface.

Previous:	The distribution used in the previous run
Uniform:	A uniform distribution, e.g. the peak pressure at each location on the surface of the transducer isa assumed to be the same
New:	The pressure distribution will be entered from the key board or a data file

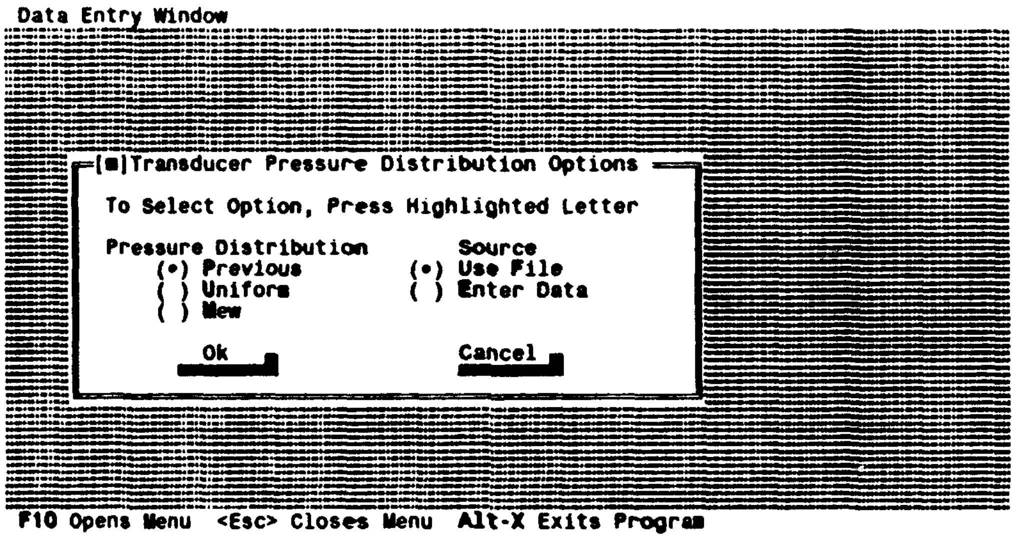

Figure A.7. Transduce Pressure Distribution Options

To select a value, either press the highlighted letter or move the mouse cursor over the option and click. The default is to use the normalized pressure distribution for the previous run. To change to uniform distribution, press U or move the mouse cursor over **Uniform** and click. To enter a new pressure distribution, press N or use the mouse cursor to click on New. If New is selected, the source choices become active. The default is to read an existing data file which describes the pressure distribution. To enter a new data file, press **E** or click on **Enter Data** with the mouse cursor. Once the final choice has been made, press **<Enter>**, **O**, or click on **Ok** with the mouse cursor.

If the source option is **Use File**, the **Pressure Distribution File Name Menu** shown in Figure A.8 appears. Press **F**, use the **Tab** key, or the mouse cursor to select **File Name**. Key in the file name and press **<Enter>**. If an error occurs during the data entry, a descriptive message will appear in the Error Message window.

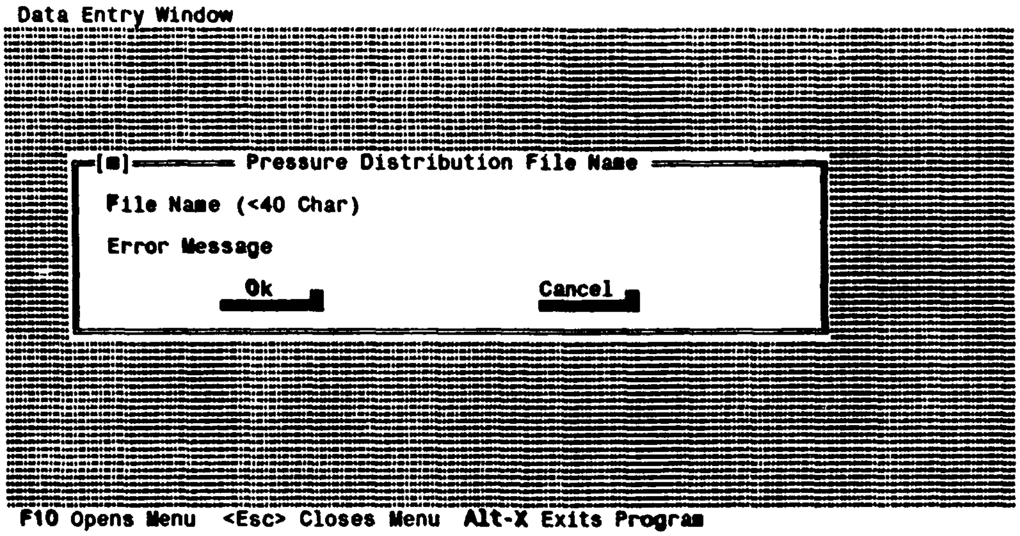

Figure A.8. Entering New File Name

Once the desired field name has been selected press **<Enter>**. To terminate the process at any time, tab to the **Cancel** button or click on this button with the mouse cursor.

A similar process is followed in generating a new data file. Select the **Enter Data** option from the menu shown in Figure A.7 and press **<Enter>**. The **New Pressure Distribution File Menu** shown in Figure A.9 appears. Press **F**, use the **Tab** key, or the mouse cursor to select **File Name**. Key in the file name. Use the **Tab** key or the mouse cursor to select **Number of points**. Key in a value between 1 and 100 and press **<Enter>**. If an error occurs during data entry, a descriptive message appears in the Error Message window. Use the **Tab** key or the mouse cursor to move to the data entry window that contains the error and key in a corrected value. To exit at any time without entering a new data file, use the **Tab** key or the mouse cursor to select the **Cancel** butt n and press enter or click on the button.

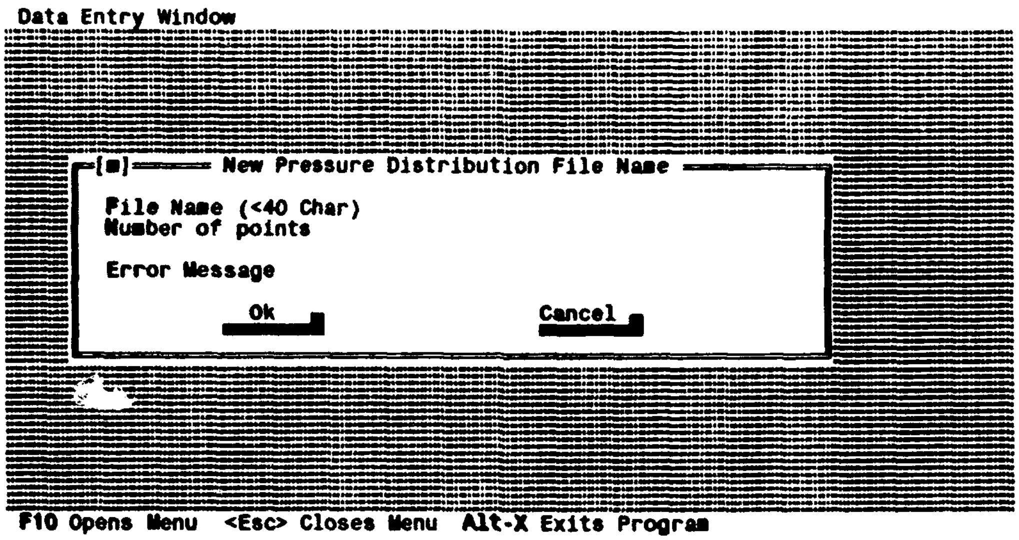

Figure A.9. Entering a new file name and number of points

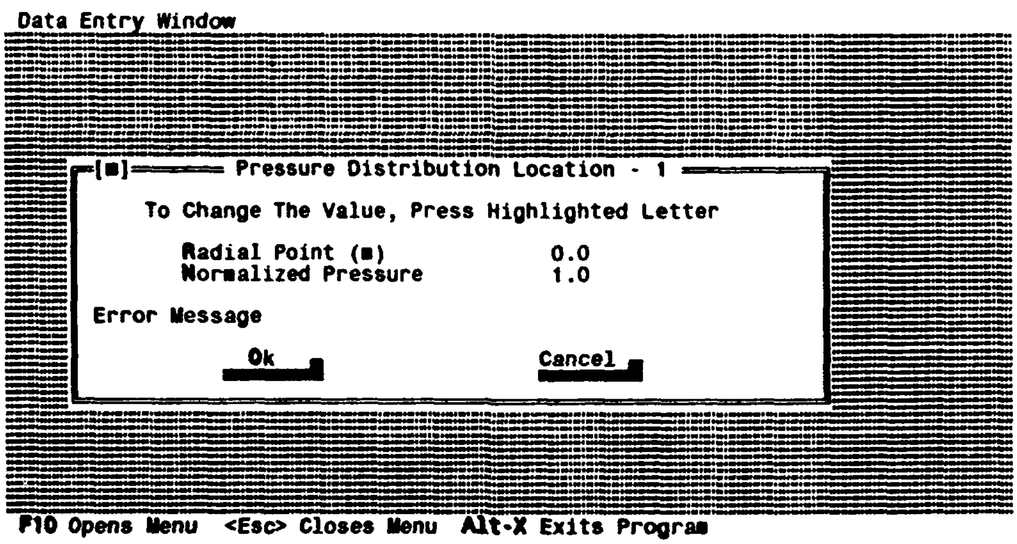

Figure A.10. Entering New Pressure Distribution File

To enter the transducer amplitude distribution use the menu shown in Figure A.10. First use the **Tab** key or the mouse to select **Radial Point (m)**. The radial location is measured normal to the transducer axis of symmetry. The first entry must be the value on the symmetry axis, e.g. 0. After keying in the appropriate value use the **Tab** key or the

mouse to select **Normalized Pressure**. Key in a value between 0 and 1. If the radial location extends beyond the transducer boundary, an error message appears. Also, a normalized pressure greater than unity generates an error message. After the location and pressure value have been correctly entered, press **<Enter>** or the **Ok** button. After the last data entry is made, the program returns to the program main menu shown in Figure A.1.

A.3.5. Program Execution

After data entry is completed, press **Alt-X** or use the mouse cursor to click the **Alt-X Exits Program** section of the menu shown in Figure A.1 to start program execution. Depending on the data values used and computer CPU speed, program execution can run from several hours to as much as a couple of days. While the program is executing, the status message shown below is written to the screen.

```
Nstep =   264 z =   0.136237 k =  29 m =   4 Imax =  113
Nstep =   273 z =   0.140900 k =  30 m =   4 Imax =  115
Nstep =   282 z =   0.145563 k =  31 m =   4 Imax =  115
```

Nstep is the calculation step number. Z is the normalized on-axis distance from the face of the transducer. K is the number of the output point. The current number of harmonic terms being used in the calculation is specified by m. Finally, Imax is the current location of floating outer boundary for the edge of the calculational mesh.

To interrupt program execution press **Ctrl-C** or **Ctrl-Break**. After two screen writing cycles, all relevant data is saved to the hard disk and normal program termination occurs. To restart the program, type **LFOCUS** and press **<Enter>**. When the program has finished running, it writes **PROGRAM EXECUTION COMPLETED** to the screen.

A.4. Output Data File Structures

After **LFOCUS.EXE** has completed execution four text data files are written - amplitude and phase files for on-axis and off-axis data points. Figure A.11 summarizes the

relevant characteristics of AMPZ and PHASEZ - the on-axis amplitude and phase spectra output files. The first 44 lines in each file summarizes the input data used to conduct the run. This is followed in each case by the amplitude spectra or phase spectra, depending upon the file type. Since the number of spectral terms will vary, depending upon the nonlinearity of problem, e.g. initial pressure, effective focusing gain, etc., the first output is the number of spectral terms, the second output is the location along the z axis. The subsequent output terms are either the amplitude or phase spectra depending upon the file type. The amplitude spectral terms are normalized to the peak pressure. The phase terms are in radians.

```
                                 Ampz. Data Structure
Lines 1 through 44  = file header
Line 45 through end = amplitude spectra with
  #         z              A[0]            A[1]            A[2]        •••
  4    5.6633E-0003    0.0000E+0000    1.0032E+0000    6.2014E-0005    •••
  4    1.0327E-0002    0.0000E+0000    1.0057E+0000    1.0892E-0004    •••
  4    1.4990E-0002    0.0000E+0000    1.0081E+0000    1.5616E-0004    •••

                                 Phasez. Data Structure
Lines 1 through 44  = file header
Line 45 through end = phase spectra with
  #         z              Phi[0]          Phi[1]          Phi[2]      •••
  4    5.6633E-0003    0.0000E+0000   -4.8870E-0006   -6.4645E-0006    •••
  4    1.0327E-0002    0.0000E+0000   -1.4608E-0005   -1.8326E-0005    •••
  4    1.4990E-0002    0.0000E+0000   -2.9689E-0005   -3.6417E-0005    •••
```

Figure A.11. Ampz and Phasez Output Data Structure

A similar file structure is used for the off-axis points, as shown in Figure A.12. Again, the first 44 lines of AMPZR or PHASEZR summarize the input data. Line nine is an integer entry which specifies the number of off-axis data output points. Starting with the first output location, a record is written which contains the number of spectral terms used at the location, the normalized on-axis location, the normalized off axis location, and the first through n^{th} amplitude or phase terms.

Ampzr. Data Structure

Lines 1 through 44 = file header
Line 9 = number of-axis data output locations
Line 45 through end = amplitude spectra with

#	z	r[1]	A[0]	A[1]	•••
5	7.5000E-0001	0.0000E+0000	0.0000E+0000	2.8336E+0000	•••
		r[2]			
5	7.5000E-0001	2.0000E-0002	0.0000E+0000	2.7830E+0000	•••
		r[3]			
5	7.5000E-0001	4.0000E-0002	0.0000E+0000	2.6362E+0000	•••

Phasezr. Data Structure

Lines 1 through 44 = file header
Line 9 = number of-axis data output locations
Line 45 through end = amplitude spectra with

#	z	r[1]	Phi[0]	Phi[1]	•••
5	7.5000E-0001	0.0000E+0000	0.0000E+0000	9.7667E+0000	•••
		r[2]			
5	7.5000E-0001	2.0000E-0002	0.0000E+0000	1.0189E+0001	•••
		r[3]			
5	7.5000E-0001	4.0000E-0002	0.0000E+0000	1.1529E+0001	•••

Figure A.12. Ampzr and Phasezr Output Data Structure

The amplitude spectra are normalized relative to the initial input pressure and the phase spectral terms are in degrees.

APPENDIX B

MENU.PAS

PROGRAM LISTING

This appendix provides a listing of the menu program used by LFOCUS.EXE. This program is written in Turbo Pascal v 6.0 and called by the FORTRAN code during initial execution. Since the program uses the standard Turbo Vision applications framework, only the unique source code is commented.

```
program MENU;

uses Objects,
     Drivers,
     Views,
     Menus,
     Dialogs,
     App;

const
  MaxLines            = 100;
  WinCount: Integer =   0;
  cmFileOpen          = 100;
  cmNewWin            = 101;
  cmNewDialog         = 102;
  cmOffAxisData       = 103;
  cmNewTransducer     = 104;
  cmRunInput          = 105;
  cmBoundary          = 106;
  cmPrevious          = 107;
  cmUniform           = 108;
  cmNewFile           = 109;
  cmNewFileValues     = 110;

var
  LineCount,
  MaxNumber,
  OffAxisIndex,
  code,
  n              : INTEGER;
  Lines: array[0..MaxLines - 1] of PString;
  errorflag      : BOOLEAN;

type

{        Record types used by the program for temporary data I/O                    }

  RunInputValues = record
    MINM              : STRING[128];
    MAXM              : STRING[128];
    Npoints           : STRING[128];
    NDpoints          : STRING[128];
    DeltaR            : STRING[128];
    Zmin              : STRING[128];
    Zmax              : STRING[128];
    Rgmax             : STRING[128];
    RadiusMax         : STRING[128];
    errormessage      : STRING[128];
  end;

  TransducerData = RECORD
    Diameter     : STRING[128];
    RadiusOfC    : STRING[128];
    Frequency    : STRING[128];
    Pressure     : STRING[128];
    errormessage : STRING[128];
  end;

  ZuData = RECORD
    location     : STRING[128];
    errormessage : STRING[128];
  end;

  BoundaryData = RECORD
    RadioButtonDataA : Word;
    RadioButtonDataB : Word;
  end;

  FileNameData = RECORD
    Name         : STRING[128];
    ErrorMessage : STRING[128];
  end;
```

```
FileNameData2 = RECORD
  Name          : STRING[128];
  NumberPoints  : STRING[128];
  ErrorMessage  : STRING[128];
end;

BoundaryDataValues = RECORD
  location      : STRING[128];
  NormalizedP   : STRING[128];
  errormessage  : STRING[128];
end;


TMyApp = object(TApplication)
  procedure HandleEvent(var Event: TEvent); virtual;
  procedure InitMenuBar; virtual;
  procedure InitStatusLine; virtual;
  procedure NewTransducer;
  procedure RunParameters;
  procedure OffAxisData;
  procedure NewWindow;
  procedure SelectBoundary;
  procedure EnterFileName;
  procedure EnterFileName2;
  procedure BoundaryAxisData;
end;

PInterior = ^TInterior;
TInterior = object(TScroller)
  constructor Init(var Bounds: TRect; AHScrollBar, AVScrollBar: PScrollBar);
  procedure Draw; virtual;
end;

PDemoWindow = ^TDemoWindow;
  TDemoWindow = object(TWindow)
    RInterior, LInterior: PInterior;
    constructor Init(Bounds: TRect; WinTitle: String; WindowNo: Word);
    function MakeInterior(Bounds: TRect; Left: Boolean): PInterior;
    procedure SizeLimits(var Min, Max: TPoint); virtual;
end;

PDemoDialog = ^TDemoDialog;
  TDemoDialog = object(TDialog)
end;

var
  Transducer             : TransducerData;
  RunInput               : RunInputValues;
  Zu                     : Array[0..50] of ZuDATA;
  BoundaryChoice         : BoundaryData;
  FileName               : FileNameData;
  FileName2              : FileNameData2;
  BoundaryChoiceValues   : Array[0..100] of BoundaryDataValues;
  NumberBoundaryPoints   : INTEGER;
  textfile               : TEXT;

constructor TInterior.Init(var Bounds: TRect; AHScrollBar, AVScrollBar: PScrollBar);
begin
  TScroller.Init(Bounds, AHScrollBar, AVScrollBar);
  Options := Options or ofFramed;
  SetLimit(128, LineCount);
end;
```

```
procedure TInterior.Draw;
var
  Color: Byte;
  I, Y: Integer;
  B: TDrawBuffer;
begin
  Color := GetColor(1);
  for Y := 0 to Size.Y - 1 do
  begin
    MoveChar(B, ' ', Color, Size.X);
    i := Delta.Y + Y;
    if (I < LineCount) and (Lines[I] <> nil) then
      MoveStr(B, Copy(Lines[I]^, Delta.X + 1, Size.X), Color);
    WriteLine(0, Y, Size.X, 1, B);
  end;
end;

constructor TDemoWindow.Init(Bounds: TRect; WinTitle: String; WindowNo: Word);
var
  S: string[3];
  R: TRect;
begin
  Str(WindowNo, S);
  TWindow.Init(Bounds, WinTitle + ' ' + S, wnNoNumber);
  GetExtent(Bounds);
  R.Assign(Bounds.A.X, Bounds.A.Y, Bounds.B.X div 2 + 1, Bounds.B.Y);
  LInterior := MakeInterior(R, True);
  LInterior^.GrowMode := gfGrowHiY;
  Insert(LInterior);
  R.Assign(Bounds.B.X div 2, Bounds.A.Y, Bounds.B.X, Bounds.B.Y);
  RInterior := MakeInterior(R,False);
  RInterior^.GrowMode := gfGrowHiX + gfGrowHiY;
  Insert(RInterior);
end;

function TDemoWindow.MakeInterior(Bounds: TRect; Left: Boolean): PInterior;
var
  HScrollBar, VScrollBar: PScrollBar;
  R: TRect;
begin
  R.Assign(Bounds.B.X-1, Bounds.A.Y+1, Bounds.B.X, Bounds.B.Y-1);
  VScrollBar := New(PScrollBar, Init(R));
  VScrollBar^.Options := VScrollBar^.Options or ofPostProcess;
  if Left then VScrollBar^.GrowMode := gfGrowHiY;
  Insert(VScrollBar);
  R.Assign(Bounds.A.X+2, Bounds.B.Y-1, Bounds.B.X-2, Bounds.B.Y);
  HScrollBar := New(PScrollBar, Init(R));
  HScrollBar^.Options := HScrollBar^.Options or ofPostProcess;
  if Left then HScrollBar^.GrowMode := gfGrowHiY + gfGrowLoY;
  Insert(HScrollBar);
  Bounds.Grow(-1,-1);
  MakeInterior := New(PInterior, Init(Bounds, HScrollBar, VScrollBar));
end;

procedure TDemoWindow.SizeLimits(var Min, Max: TPoint);
var R: TRect;
begin
  TWindow.SizeLimits(Min, Max);
  Min.X := LInterior^.Size.X + 9;
end;
```

```
{       This is the event handler used to call the data entry procedures.
        cmNewTransducer        =        call to NewTransducer
        cmRunInput             =        call to RunParamcters
        cmOffAxisData          =        call to OffAxisData
        cmNewWin               =        call to NewWindow
        cmBoundary             =        call to SelectBoundary                    }

procedure TMyApp.HandleEvent(var Event: TEvent);
begin
  TApplication.HandleEvent(Event);
  if Event.What = evCommand then
  begin
    case Event.Command of
      cmNewWin          : NewWindow;
      cmNewTransducer   : NewTransducer;
      cmRunInput        : RunParameters;
      cmOffAxisData     : begin
        VAL(RunInput.NDpoints,MaxNumber,Code);
        For OffAxisIndex := 1 To MaxNumber Do OffAxisData;
      end;
      cmBoundary        : begin
        SelectBoundary;
        Case BoundaryChoice.RadioButtonDataA Of
          0 :;
          1 : begin

              {This section sets a uniform aperture distribution}

              NumberBoundaryPoints               := 1;
              BoundaryChoiceValues[0].location    := '0';
              BoundaryChoiceValues[0].NormalizedP := '1.0';
              end;
          2 : Case BoundaryChoice.RadioButtonDataB OF
                0 : begin
                    EnterFileName;
                    IF skipflag = FALSE THEN
                    begin
                    Assign(textfile,FileName.name);
                    RESET(textfile);
                    READLN(textfile,NumberBoundaryPoints);
                    FOR n := 0 TO NumberBoundaryPoints - 1 DO
                      begin
                        READLN(textfile,BoundaryChoiceValues[n].location);
                        READLN(textfile,BoundaryChoiceValues[n].NormalizedP);
                        BoundaryChoiceValues[n].errormessage := '';
                      end;
                    FOR n := NumberBoundaryPoints TO 100 DO
                      WITH  BoundaryChoiceValues[n] DO
                        begin
                          location     := '';
                          NormalizedP  := '';
                          errormessage := '';
                        end;
                   CLOSE(textfile);
                   end;
                  end;
                1 : begin
                    EnterFileName2;
                    IF skipflag = FALSE THEN
                    begin
                    VAL(FileName2.NumberPoints,NumberBoundaryPoints,Code);
                    For OffAxisIndex := 0 to NumberBoundaryPoints - 1 Do BoundaryAxisData;
                    end;
                  end;
                end;
        end;
      end
    else
      Exit;
    end;
    ClearEvent(Event);
  end;
end;
```

{ This is the interface for the pull-down menu system. It connects the commands with the keystrokes that activate each individual command. For example, R or F6 are linked to cmRunInput which is used to call the procedure **RunParameters**. }

```
procedure TMyApp.InitMenuBar;
var R: TRect;
begin
  GetExtent(R);
  R.B.Y := R.A.Y + 1;
  MenuBar := New(PMenuBar, Init(R, NewMenu(
    NewSubMenu('Data Entry ~W~indow', hcNoContext, NewMenu(
      NewItem('~R~un Parameters','F6',kbF6, cmRunInput, hcNoContext,
      NewItem('~O~ff-Axis Data Points','F7',kbF7, cmOffAxisData, hcNoContext,
      NewItem('~T~ransducer', 'F8', kbF8, cmNewTransducer, hcNoContext,
      NewItem('~B~oundary Values', 'F9', kbF9, cmBoundary, hcNoContext,
      nil))))),
    nil))
  ));
end;
```

{ This procedure controls the status line at the bottom of the main menu. It links the keys that open and close the main menu. }

```
procedure TMyApp.InitStatusLine;
var R: TRect;
begin
  GetExtent(R);
  R.A.Y := R.B.Y - 1;
  StatusLine := New(PStatusLine, Init(R,
    NewStatusDef(0, $FFFF,
      NewStatusKey('~F10~ Opens Menu  <Esc> Closes Menu', kbF10, cmMenu,
      NewStatusKey('~Alt-X~ Exits Program', kbAltX, cmQuit,
      nil)),
    nil)
  ));
end;
```

{ This procedure is used to check real input values to determine whether they are valid. If the data entry is not a real number, the error flag is set to FALSE, the error mesage is set to the string error1, and the procedure branches to the exit. If the data type is correct, the procedure checks to see whether the data entry is between the upper (UpperLimit) and lower (LowerLimit) limits. If it is not, the error flag is set to FALSE, the error message is set to the string error2, and the procedure branches to the exit. }

```
procedure CheckRealData(VAR NewVAlue, NewValue2 : STRING;
                            error1, error2 :STRING;
                        VAR errorflag : BOOLEAN;
                        LowerLimit, UpperLimit : SINGLE);

   LABEL localout;
   VAR realvalue : SINGLE;

   begin

   Val(NewVAlue,realvalue,code);
   NewValue2 := '';
   IF code <> 0 THEN
     begin
       NewValue2 := error1;
       Errorflag := FALSE;
       GOTO localout;
     end;
   IF (realvalue < lowerlimit) OR (realvalue > upperlimit) THEN
   begin
     Errorflag := FALSE;
     NewValue2 := error2;
     GOTO localout;
   end;
   localout :;
   end;
```

{ This procedure is used to check integer input values to determine whether they are valid. If the data entry is not an integer, the error flag is set to FALSE, the error mesage is set to the string error1, and the procedure branches to the exit. If the data type is correct, the procedure checks to see whether the data entry is between the upper (UpperLimit) and lower (LowerLimit) limits. If it is not, the error flag is set to FALSE, the error message is set to the string error2, and the procedure branches to the exit. }

```
procedure CheckIntegerData(VAR NewVAlue, NewValue2 : STRING;
                                  error1, error2 :STRING;
                              VAR errorflag : BOOLEAN;
                              LowerLimit, UpperLimit : SINGLE);

   LABEL localout;
   VAR integervalue : INTEGER;

   begin

   Val(NewVAlue,integervalue,code);
   NewValue2 := '';
   IF code <> 0 THEN
     begin
       NewValue2 := error1;
       Errorflag := FALSE;
       GOTO localout;
     end;
   IF (integervalue < lowerlimit) OR (integervalue > upperlimit) THEN
   begin
     Errorflag := FALSE;
     NewValue2 := error2;
     GOTO localout;
   end;
   localout :
   end;
```

{ This procedure checks to make sure that the filename conforms to PC-DOS standards }

```
procedure CheckFileName(VAR name,errormessage:STRING);

  LABEL localout;

  begin {CheckFileName}

  errorflag := TRUE;
  errormessage := '';

  IF length(name)>12 Then
    begin
      Errorflag := FALSE;
      Errormessage := 'Length > 12 Characters!';
      GOTO localout;
    end;

  IF ((POS('.',name) > 9) or (POS('.',name) = 0)) and (length(name) >= 9) THEN
    begin
      Errorflag := FALSE;
      Errormessage := 'Name > 8 Characters!';
      GOTO localout;
    end;

  IF (Length(name) - POS('.',name)) > 3 THEN
    begin
      Errorflag := FALSE;
      Errormessage := 'Extension > 8 Characters!';
      GOTO localout;
    end;

  localout :;

  end;  {CheckFileName}
```

```
{       This procedure reads the input parameters for the run.                                    }

procedure TMyApp.RunParameters;
var
  Thomas: PView;
  Dialog: PDemoDialog;
  R: TRect;
  C: Word;
  errormessage,
  tempstring : STRING;
  yfirst,
  xleft,
  xright,
  xleft2,
  xright2,
  xleft3,
  xright3 : BYTE;
  lowervalue : INTEGER;

procedure CheckRunParameters;

{       This is the local procedure that checks the run parameter data entries.  It uses CheckIntegerData and
        CheckRealData with the appropriate values.                                                }

VAR
   newcode,
   minimumM,
   maximumM : INTEGER;
   UpperBound : SINGLE;

LABEL Done;

begin

{       First set the errorflag                                                                   }

    RunInput.errormessage := '';
    errorflag := TRUE;

{       Then check each of the record values starting with the minimum number of spectral terms, Minm. If an error occurs
        in any of the checks, the error flag is set to FALSE before exiting the procedure.       }

    CheckIntegerData(RunInput.Minm,RunInput.errormessage,
    'Minimum Spectral Term Not Integer!',
    'Minimum Spectral Term Out Of Range!',errorflag, 3, 48);
    IF errorflag = FALSE THEN GOTO Done;

    VAL(RunInput.Minm,MinimumM,code);

    CheckIntegerData(RunInput.Maxm,RunInput.errormessage,
    'Maximum Spectral Term Not Integer!',
    'Maximum Spectral Term Out Of Range!',errorflag, MinimumM, 51);
    IF errorflag = FALSE THEN GOTO Done;

    CheckIntegerData(RunInput.Npoints,RunInput.errormessage,
    'Number Of On-Axis Points Not An Integer!',
    'Number Of On-Axis Points Out Of Range!',errorflag, 99, 401);
    IF errorflag = FALSE THEN GOTO Done;

    CheckIntegerData(RunInput.NDpoints,RunInput.errormessage,
    'Number Of Off-Axis Points Not An Integer!',
    'Number Of Off-Axis Points Out Of Range!',errorflag, 0, 41);
    IF errorflag = FALSE THEN GOTO Done;

    CheckRealData(RunInput.DeltaR,RunInput.errormessage,
    'Radial Step Size Not A Number!',
    'Radial Step Size Out Of Range!',errorflag, 0.001, 0.0201);
    IF errorflag = FALSE THEN GOTO Done;

    CheckRealData(RunInput.Zmin,RunInput.errormessage,
    'Minimum On-Axis Output Not A Number!',
    'Minimum On-Axis Output Of Range!',errorflag, 0.001, 0.99);
    IF errorflag = FALSE THEN GOTO Done;
```

```
      CheckRealData(RunInput.Zmax,RunInput.errormessage,
      'Maximum On-Axis Output Not A Number!',
      'Maximum On-Axis Output Of Range!',errorflag, 1.05, 2.00);
      IF errorflag = FALSE THEN GOTO Done;

      CheckRealData(RunInput.Rgmax,RunInput.errormessage,
      'Maximum Off-Axis Range Not A Number!',
      'Maximum Off-Axis Range Out Of Range!',errorflag, 1.1, 10.0);
      IF errorflag = FALSE THEN GOTO Done;

      CheckIntegerData(RunInput.RadiusMax,RunInput.errormessage,
      'Maximum Off-Axis Points Not A Number!',
      'Maximum Off-Axis Points Out Of Range!',errorflag, 10, 200);
      IF errorflag = FALSE THEN GOTO Done;
    done:
  end;

  begin
    Xleft    := 4;
    Xright   := Xleft + 40;
    Xleft2   := Xright + 1;
    Xright2  := Xleft2 + 8;
    Yfirst   := 4;
    R.Assign(15, 1, 75, yfirst + 16);
    TEmpString := 'Maximum # Spectra      (Min # to 50):';
    Dialog := New(PDemoDialog, Init(R, 'RunParameters'));
    with Dialog^ do
    begin
      R.Assign(xleft, 2, xleft + 48, 3);
      Insert(New(PLabel, Init(R, 'Use Tab key or mouse to select data entry field', Thomas)));
      R.Assign(xleft2, yfirst, xright2, yfirst + 1);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(Xleft,  yfirst, Xright, yfirst + 1);
      Insert(New(Plabel,Init(R,'Minimum # Spectra         (4 to 49):',Thomas)));
      R.Assign(xleft2, yfirst + 1, xright2, yfirst + 2);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 1, xright, yfirst + 2);
      Insert(New(Plabel,Init(R,TempString,Thomas)));
      R.Assign(xleft2, yfirst + 2, xright2, yfirst + 3);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 2, xright, yfirst + 3);
      Insert(New(Plabel,Init(R,'Number On-Axis Points     (100 to 400):',Thomas)));
      R.Assign(xleft2, yfirst + 3, xright2, yfirst + 4);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 3, xright, yfirst + 4);
      Insert(New(Plabel,Init(R,'Number Off-Axis Points      (1 to 40):',Thomas)));
      R.Assign(xleft2, yfirst + 4, xright2, yfirst + 5);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 4, xright, yfirst + 5);
      Insert(New(Plabel,Init(R,'Radial Step Size      (0.02 to 0.001):',Thomas)));
      R.Assign(xleft2, yfirst + 5, xright2, yfirst + 6);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 5, xright, yfirst + 6);
      Insert(New(Plabel,Init(R,'Min On-Axis Output     (0.001 to 0.99):',Thomas)));
      R.Assign(xleft2, yfirst + 6, xright2, yfirst + 7);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 6, xright, yfirst + 7);
      Insert(New(Plabel,Init(R,'Max On-Axis Output      (1.05 to 2.0):',Thomas)));
      R.Assign(xleft2, yfirst + 7, xright2, yfirst + 8);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 7, xright, yfirst + 8);
      Insert(New(Plabel,Init(R,'Max Off-Axis Range      (1.1 to 10.0):',Thomas)));
      R.Assign(xleft2, yfirst + 8, xright2, yfirst + 9);
      Thomas := New(PInputLine, Init(R,128));
      Insert(Thomas);
      R.Assign(xleft,  yfirst + 8, xright, yfirst + 9);
      Insert(New(Plabel,Init(R,'Max Off-Axis Points       (10 to 200):',Thomas)));
      R.Assign(16, yfirst + 10, 56, yfirst + 11);
      Thomas := New(PinputLine, Init(R,128));
```

```
    Insert(Thomas);
    R.Assign(1,  yfirst + 10, 15, yfirst + 11);
    Insert(New(Plabel,Init(R,'Error Message',Thomas)));
```

{ These are the escape buttons from the box. To use the values entered execute cmOK. To discard the changes execute cmCancel. }

```
    R.Assign(xleft + 9,  yfirst + 12, xleft + 19, yfirst + 14);
    Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    R.Assign(xleft + 34, yfirst + 12  xleft + 44, yfirst + 14);
    Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  end;
  REPEAT
  Dialog^.SetData(RunInput);
  C := DeskTop^.ExecView(Dialog);
  if C <> cmCancel then Dialog^.GetData(RunInput);
```

{ Check to see whether there are any errors. If an error has occurred, then do not exit the program. }

```
  CheckRunParameters;
  UNTIL ((C = cmCancel) or (C = cmOK)) and (errorflag = TRUE);
  Dispose(Dialog, Done);
end;
```

{ This procedure is used to input or change transducer related parameters, such as the radius of curvature, diameter, initial on-axis power, and frequency. }

```
procedure TMyApp.NewTransducer;
var
  Thomas2: PView;
  Dialog2: PDemoDialog;
  R: TRect;
  C: Word;
  errormessage : STRING;
  xleft,
  xright,
  xleft2,
  xright2,
  yfirst   : BYTE;
```

{ This is the local procedure that checks the transducer data entries. It uses CheckIntegerData and CheckRealData with the appropriate values. }

```
Procedure CheckTransducer;

  LABEL Done;

  begin {CheckTransducer}
```

{ First set the error message and error flag. Then check each of the data entries starting with the diameter. If an error occurs, set the error flag to FALSE and exit. }

```
    transducer.errormessage := '';
    errorflag := TRUE;
    CheckRealData(transducer.diameter,transducer.errormessage,
    'Diameter Entry Is Not A Number!',
    'Diameter Entry Out Of Range!',errorflag, 0, 1);
    IF errorflag = FALSE THEN GOTO Done;

    CheckRealData(transducer.RadiusOfC,transducer.errormessage,
    'Focal Length Is Not A Number!',
    'Focal Length Out Of Range!',errorflag, 0, 1);
    IF errorflag = FALSE THEN GOTO Done;

    CheckRealData(transducer.frequency,transducer.errormessage,
    'Frequency Is Not A Number!',
    'Frequency Out Of Range!',errorflag, 0, 10);
    IF errorflag = FALSE THEN GOTO Done;

    CheckRealData(transducer.pressure,transducer.errormessage,
    'Pressure Is Not A Number!',
    'Pressure Out Of Range!',errorflag, 0, 2000);

    done :
  end;  {CheckTransducer}

begin
  Yfirst := 2;
  Xleft := 2;
  Xright := Xleft + 34;
  Xleft2 := Xright + 1;
  Xright2 := Xleft2 + 10;
  R.Assign(5, 6, 58, 20);
  Dialog2 := New(PDemoDialog, Init(R, 'Transducer Characteristics'));
  with Dialog2^ do
  begin
    R.Assign(Xleft, Yfirst, 50, Yfirst + 1);
    Insert(New(PLabel, Init(R, 'Use Tab key or mouse to select data entry field', Thomas2)));
    R.Assign(Xleft2, Yfirst + 2, Xright2, Yfirst + 3);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst + 2, Xright, Yfirst + 3);
    Insert(New(Plabel,Init(R,'Diameter          (Meters: 0 .. 1):',Thomas2)));
    R.Assign(Xleft2, Yfirst + 3, Xright2, Yfirst + 4);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(xleft, yfirst + 3, xright, Yfirst + 4);
    Insert(New(Plabel,Init(R,'Focal Length      (Meters: 0 .. 1):',Thomas2)));
    R.Assign(Xleft2, Yfirst + 4, Xright2, Yfirst + 5);
```

```
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(xleft, yfirst +4, xright, Yfirst + 5);
    Insert(New(Plabel,Init(R,'Initial Frequency  (MHz: 0 .. 10):',Thomas2)));
    R.Assign(Xleft2, Yfirst + 5, Xright2, Yfirst + 6);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(xleft, yfirst + 5, xright, Yfirst + 6);
    Insert(New(Plabel,Init(R,'Initial Pressure (kPa: 0 .. 2000):',Thomas2)));
    R.Assign(16,Yfirst + 7, 49, Yfirst + 8);
    Thomas2 := New(PinputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(1,Yfirst + 7, 15,Yfirst + 8);
    Insert(New(Plabel,Init(R,'Error Message',Thomas2)));

{       These are the escape buttons from the box. To use the values entered execute cmOK. To discard the changes
        execute cmCancel.                                                                                }

    R.Assign(10, Yfirst + 9, 20, Yfirst + 11);
    Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    R.Assign(35, Yfirst + 9, 45, Yfirst + 11);
    Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  end;
  REPEAT
    Dialog2^.SetData(Transducer);
    C := DeskTop^.ExecView(Dialog2);
    if C <> cmCancel then Dialog2^.GetData(Transducer);

{       Check to see whether there are any errors. If an error has occurred, then do not exit the program.   }

    CheckTransducer;
  UNTIL (((C = cmCancel) or (C = cmOK)) and (errorflag = TRUE));
  Dispose(Dialog2,Done);
end;
```

```
{       This procedure is used to select the type and source of the initial pressure distribution at the face of the transducer.
        The options allow the user to enter the data or select a file.                                                          }

procedure TMyApp.SelectBoundary;
var
  Thomas2: PView;
  Dialog2: PDemoDialog;
  R: TRect;
  C: Word;
  errormessage : STRING;
  xleft,
  xright,
  xleft2,
  xright2,
  XleftB1,
  YleftB1,
  XRightB1,
  YRightB1,
  XleftB2,
  XRightB2,
  YLeftB2,
  YRightB2,
  yfirst    : BYTE;
  textfile : TEXT;


begin
  Yfirst := 2;
  Xleft := 2;
  XleftB1 := 6;
  YleftB1 := 5;
  XRightB1 := XleftB1 + 14;
  YRightB1 := YleftB1 + 3;
  XleftB2 := XRightB1 + 6;
  YleftB2 := YleftB1;
  XRightB2 := XLeftB2 + 18;
  YRightB2 := YleftB2 + 2;
  Xright := Xleft + 34;
  Xleft2 := Xright + 1;
  Xright2 := Xleft2 + 10;
  R.Assign(5, 6, 55, 18);
  Dialog2 := New(PDemoDialog, Init(R, 'Transducer Pressure Distribution Options'));
  with Dialog2^ do
  begin
    R.Assign(Xleft, Yfirst, 46, Yfirst + 1);
    Insert(New(PLabel, Init(R, 'To Select Option, Press Highlighted Letter', Thomas2)));

{       This set of radio buttons allow the selection of the values used in a previous run, a uniform distribution, or a set of
        dta values contained in a file.                                                                                        }

    R.Assign(XleftB1,YleftB1,XRightB1,YRightB1);
    Thomas2 := New(PRadioButtons,Init(R,
              NewSItem('~P~revious',
              NewSItem('~U~niform',
              NewSItem('~N~ew',
              nil)))
              ));
    Insert(Thomas2);

    R.Assign(XleftB1-4, YleftB1 - 1, XRightB1+4, YLeftB1);
    Insert(New(PLabel, Init(R, 'Pressure Distribution', Thomas2)));

{       This allows the selection of a previous file or the creation of new file.                                               }

    R.Assign(XleftB2,YleftB2,XRightB2,YRightB2);
    Thomas2 := New(PRadioButtons,Init(R,
              NewSItem('Use ~F~ile',
              NewSItem('~E~nter Data',
              nil))
              ));
    Insert(Thomas2);

    R.Assign(XleftB2 + 4, YleftB2- 1, XRightB2, YleftB2);
    Insert(New(PLabel, Init(R, 'Source', Thomas2)));
```

{ These are the escape buttons from the box. To use the values entered execute cmOK. To discard the changes execute cmCancel. }

```
    R.Assign(7, Yfirst + 7, 17, Yfirst + 9);
    Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    R.Assign(29, Yfirst + 7, 39, Yfirst + 9);
    Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  end;
  REPEAT
    Dialog2^.SetData(BoundaryChoice);
    C := DeskTop^.ExecView(Dialog2);
    if C <> cmCancel THEN Dialog2^.GetData(BoundaryChoice);
  UNTIL (((C = cmCancel) or (C = cmOK)));
  Dispose(Dialog2,Done);
end;
```

{ This procedure is used to enter and check the name assoicated with an existing data file. }

```
procedure TMyApp.EnterFileName;
var
  Thomas2: PView;
  Dialog2: PDemoDialog;
  R: TRect;
  C: Word;
  errormessage : STRING;
  xleft,
  xright,
  xleft2,
  xright2,
  ButtonLength,
  OKXlocation,
  OKYlocation,
  CXlocation,
  CYlocation,
  yfirst   : BYTE;
  textfile : TEXT;
  DirInfo : SearchRec;

begin {EnterFileName}
  Xleft := 2;
  Xright := Xleft + 22;
  Yfirst := 2;
  Xleft2 := Xright + 2;
  Xright2 := Xleft2 + 30;
  ButtonLength := 10;
  OKXlocation  := 10;
  CXlocation   := 35;
  OKYlocation  := 6;
  CYlocation   := OKYlocation;
  R.Assign(5, 6, 65, 15);
  Dialog2 := New(PDemoDialog, Init(R, 'Pressure Distribution File Name'));
  with Dialog2^ do
  begin
    R.Assign(Xleft2, Yfirst, Xright2, Yfirst+1);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst, Xright, Yfirst + 1);
    Insert(New(Plabel,Init(R,'~F~ile Name (<40 Char)',Thomas2)));
    R.Assign(Xleft2, Yfirst+2, Xright2, Yfirst+3);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst+2, Xright, Yfirst + 3);
    Insert(New(Plabel,Init(R,'Error Message',Thomas2)));
```

{ These are the escape buttons from the box. To use the values entered execute cmOK. To discard the changes execute cmCancel. }

```
    R.Assign(OKXlocation, OKYLocation, OKXlocation + ButtonLength, OKYlocation + 2);
    Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    R.Assign(CXlocation, CYlocation, CXlocation + ButtonLength, CYlocation + 2);
    Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  end;
  REPEAT
    Dialog2^.SetData(FileName);
    C := DeskTop^.ExecView(Dialog2);
    if C <> cmCancel then
    begin
      skipflag := FALSE;
      Dialog2^.GetData(FileName);
    checkfilename(filename.name,filename.errormessage);
    IF (C = cmOK) and (FileName.Name = '') THEN errorflag := FALSE
                                          ELSE
    IF Errorflag <> FALSE THEN
    begin
    FindFirst(filename.name,AnyFile,DirInfo);
    IF DosError <> 0 THEN
      begin
      errorflag := FALSE;
      filename.errormessage := 'File '+filename.name+ ' Not Found!';
      end;
```

```
      end;
      end               else
      begin
        errorflag := TRUE;
        skipflag  := TRUE;
      end;

    UNTIL (((C = cmCancel) or (C = cmOK)) and (errorflag = TRUE));
    IF C = cmCancel THEN BoundaryChoice.RadioButtonDataA := 0;
    Dispose(Dialog2,Done);

  end;  {EnterFileName}
```

{ This procedure is used to enter the name of a data file which is to be created, the number of data points which will be contained in the file, and check the file name to make sure it conforms ot DOS standards. }

```
procedure TMyApp.EnterFileName2;
var
  Thomas2: PView;
  Dialog2: PDemoDialog;
  R: TRect;
  C: Word;
  errormessage : STRING;
  xleft,
  xright,
  xleft2,
  xright2,
  ButtonLength,
  OKXlocation,
  OKYlocation,
  CXlocation,
  CYlocation,
  yfirst   : BYTE;
  textfile : TEXT;

Procedure CheckNumberPoints;
```

{ This procedure checks the number of points to make sure it is an integer and within range. }

```
  LABEL Done;

  begin {CheckNumberPoints}

    filename2.errormessage := '';
    errorflag := TRUE;
    CheckIntegerData(filename2.NumberPoints,filename2.errormessage,
    'Number Of Points Must Be > 0',
    'Number Of Points Out Of Range!',errorflag, 0, 100);
  end;  {CheckNumberPoints}

begin {EnterFileName2}
  Xleft := 2;
  Xright := Xleft + 22;
  Yfirst := 2;
  Xleft2 := Xright + 2;
  Xright2 := Xleft2 + 30;
  ButtonLength := 10;
  OKXlocation  := 10;
  CXlocation   := 35;
  OKYlocation  := 7;
  CYlocation   := OKYlocation;
  R.Assign(5, 6, 65, 16);
  Dialog2 := New(PDemoDialog, Init(R, 'New Pressure Distribution File Name'));
  with Dialog2^ do
  begin
    R.Assign(Xleft2, Yfirst, Xright2, Yfirst+1);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst, Xright, Yfirst + 1);
    Insert(New(Plabel,Init(R,'~F~ile Name (<40 Char)',Thomas2)));
    R.Assign(Xleft2, Yfirst+1, Xright2, Yfirst+2);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst+1, Xright, Yfirst + 2);
    Insert(New(Plabel,Init(R,'~N~umber of points',Thomas2)));

    R.Assign(Xleft2, Yfirst+3, Xright2, Yfirst+4);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst+3, Xright, Yfirst + 4);
    Insert(New(Plabel,Init(R,'Error Message',Thomas2)));
```

{ These are the escape buttons from the box. To use the values entered execute cmOK. To discard the changes execute cmCancel. }

```
    R.Assign(OKXlocation, OKYLocation, OKXlocation + ButtonLength, OKYlocation + 2);
    Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    R.Assign(CXlocation, CYlocation, CXlocation + ButtonLength, CYlocation + 2);
    Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));
```

```
  end;
  REPEAT
    skipflag := FALSE;
    Dialog2^.SetData(FileName2);
    C := DeskTop^.ExecView(Dialog2);
    if C <> cmCancel then
      begin
        Dialog2^.GetData(FileName2);
        checkfilename(filename2.name,filename2.errormessage);

        IF ((C = cmOK) and (FileName2.Name = '')) THEN
          begin
            errorflag := FALSE;
            filename2.errormessage := 'No File Name Entered!';
          end
                                                        ELSE
          IF errorflag = TRUE THEN CheckNumberPoints;
      end           else
      begin
        errorflag := TRUE;
        skipflag  := true;
      end;
  UNTIL ((C = cmCancel) or ((C = cmOK) and (errorflag = TRUE)));
  IF C = cmCancel THEN BoundaryChoice.RadioButtonDataA := 0;
  Dispose(Dialog2,Done);

end;  {EnterFileName2}
```

{ This procedure is used for the off-axis data entry. }

```
procedure TMyApp.OffAxisData;
var
  Thomas2: PView;
  Dialog2: PDemoDialog;
  R: TRect;
  C: Word;
  tempval,
  errormessage : STRING;
  xleft,
  xright,
  xleft2,
  xright2,
  yfirst   : BYTE;
```

{ This is the local procedure that checks the transducer data entries. It uses CheckIntegerData and CheckRealData with the appropriate values. }

```
Procedure CheckOffAxisData;

  VAR
    minval,
    maxval :SINGLE;

  LABEL Done;

  begin {CheckOffAxisData}
```

{ First set the error message and error flag. Then check each data entry. Since this procedure is called each time the off axis index changes. If an error occurs. set the error flag to FALSE and exit. }

```
    ZU[offaxisindex].errormessage := '';
    errorflag := TRUE;
    VAL(ZU[offaxisindex -1].location,minval,code);
    VAL(RunInput.Zmax,maxval,code);
    CheckRealData(ZU[offaxisindex].location,ZU[offaxisindex].errormessage,
    'Off-Axis Entry Is Not A Number!', 'Previous Entry '+
    ZU[offaxisindex - 1].location + ' Entry Out Of Range!',errorflag, minval, maxval);
    IF errorflag = FALSE THEN GOTO Done;
    done :

  end;  {CheckOffAxisData}

begin
  Yfirst := 2;
  Xleft := 5;
  Xright := Xleft + 38;
  Xleft2 := Xright + 1;
  Xright2 := Xleft2 + 10;
  STR(offaxisindex,tempval);
  R.Assign(5, 6, 65,17);
  Dialog2 := New(PDemoDialog, Init(R, 'Off Axis Location '+ tempval));
  with Dialog2^ do
  begin
    R.Assign(Xleft, Yfirst, Xleft + 50, Yfirst + 1);
    Insert(New(PLabel, Init(R, 'Use Tab key or mouse to select data entry field', Thomas2)));
    R.Assign(Xleft2, Yfirst + 2, Xright2, Yfirst + 3);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst + 2, Xright, Yfirst + 3);
    Insert(New(Plabel,Init(R,'Output Point (> previous value < '+ Runinput.Zmax +'):',Thomas2)));
    R.Assign(16,Yfirst + 4,59, Yfirst + 5);
    Thomas2 := New(PinputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(1,Yfirst + 4,15,Yfirst + 5);
    Insert(New(Plabel,Init(R,'Error Message',Thomas2)));
```

{ These are the escape buttons from the box. To use the values entered execute cmOK. To discard the changes execute cmCancel. }

```
    R.Assign(10, Yfirst + 6, 20, Yfirst + 8);
    Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    R.Assign(35, Yfirst + 6, 45, Yfirst + 8);
    Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));
  end;
```

```
REPEAT
  Dialog2^.SetData(ZU[OffAxisIndex]);
  C := DeskTop^.ExecView(Dialog2);
  if C <> cmCancel then Dialog2^.GetData(ZU[OffAxisIndex]);

{       Check to see whether there are any errors.  If an error has occurred, then do not exit the program.       }

  CheckOffAxisData;
UNTIL (((C = cmCancel) or (C = cmOK)) and (errorflag = TRUE));
Dispose(Dialog2,Done);
end;
```

{ This procedure checks each location and pressure pair to make sure they are within range. }

```
procedure TMyApp.BoundaryAxisData;
  VAR
    Thomas2: PView;
    Dialog2: PDemoDialog;
    R: TRect;
    C: WORD;
    tempval,
    errormessage : STRING;
    xleft,
    xright,
    xleft2,
    xright2,
    yfirst   : BYTE;

  procedure CheckBoundaryAxisData;
```

{ This procedure checks the normalized pressure or the radial input point to assure that the normalized pressure is less than 1.0 and that the radial input point is less than the transducer's transverse radius. }

```
    VAR
      minval,
      maxval :SINGLE;

      LABEL Done;

  begin {CheckBoundaryAxisData}

    BoundaryChoiceValues[offaxisindex].errormessage := '';
    errorflag := TRUE;
    minval := 0;
    VAL(Transducer.Diameter,maxval,code);
    maxval := maxval/2.0;
    CheckRealData(BoundaryChoiceValues[offaxisindex].location,
    BoundaryChoiceValues[offaxisindex].errormessage,
      'Off-Axis Location Entry Is Not A Number!',
      'Location ' + BoundaryChoiceValues[offaxisindex].location+
      ' Out Of Range!', errorflag, minval, maxval);
    IF errorflag = FALSE THEN GOTO Done;

    minval := 0;
    maxval := 1;
    CheckRealData(BoundaryChoiceValues[offaxisindex].NormalizedP,
    BoundaryChoiceValues[offaxisindex].errormessage,
    'Pressure Is Not A Number!',
    'Pressure Entry Out Of Range!', errorflag, minval, maxval);
    IF errorflag = FALSE THEN GOTO Done;

    done :
  end;  {CheckBoundaryAxisData}

begin
  Yfirst := 2;
  Xleft := 8;
  Xright := Xleft + 28;
  Xleft2 := Xright + 1;
  Xright2 := Xleft2 + 10;
  STR(offaxisindex + 1,tempval);
  R.Assign(5, 6, 65,18);
  Dialog2 := New(PDemoDialog, Init(R, 'Pressure Distribution Location - '+ tempval));
  WITH Dialog2^ DO
  begin
    R.Assign(Xleft-3, Yfirst, Xleft + 50, Yfirst + 1);
    Insert(New(PLabel, Init(R, 'To Change The Value, Press Highlighted Letter', Thomas2)));
    R.Assign(Xleft2, Yfirst + 2, Xright2, Yfirst + 3);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(Xleft, Yfirst + 2, Xright, Yfirst + 3);
    Insert(New(Plabel,Init(R,'Radial ~P~oint (m)',Thomas2)));
    R.Assign(Xleft2, Yfirst + 3, Xright2, Yfirst + 4);
    Thomas2 := New(PInputLine, Init(R,128));
    Insert(Thomas2);
```

```
    R.Assign(Xleft, Yfirst + 3, Xright, Yfirst + 4);
    Insert(New(Plabel,Init(R,'~N~ormalized Pressure',Thomas2)));
    R.Assign(16,Yfirst + 5,59, Yfirst + 6);
    Thomas2 := New(PinputLine, Init(R,128));
    Insert(Thomas2);
    R.Assign(1,Yfirst + 5,15,Yfirst + 6);
    Insert(New(Plabel,Init(R,'Error Message',Thomas2)));
```

{ These are the escape buttons from the box. To use the values entered execute cmOK. To discard the changes execute cmCancel. }

```
    R.Assign(10, Yfirst + 7, 20, Yfirst + 9);
    Insert(New(PButton, Init(R, '~O~k', cmOK, bfDefault)));
    R.Assign(35, Yfirst + 7, 45, Yfirst + 9);
    Insert(New(PButton, Init(R, 'Cancel', cmCancel, bfNormal)));

  end;
  REPEAT
    Dialog2^.SetData(BoundaryChoiceValues[OffAxisIndex]);
    C := DeskTop^.ExecView(Dialog2);
    IF C <> cmCancel THEN
      Dialog2^.GetData(BoundaryChoiceValues[OffAxisIndex]);
    CheckBoundaryAxisData;
  UNTIL (((C = cmCancel) OR (C = cmOK)) AND (errorflag = TRUE));
  Dispose(Dialog2,Done);
end;
```

```
procedure TMyApp.NewWindow;
var
  Window: PDemoWindow;
  R: TRect;
begin
  Inc(WinCount);
  R.Assign(0, 0, 45, 13);
  R.Move(Random(34), Random(11));
  Window := New(PDemoWindow, Init(R, 'Demo Window', WinCount));
  DeskTop^.Insert(Window);
end;

var
  MyApp: TMyApp;
```

{ This procedure is used to initialize the radio button choices and the file names. }

```
procedure InitiateBoundary_FileName;

begin
  WITH BoundaryChoice DO
  begin
    RadioButtonDataA := 0;
    RadioButtonDataB := 0;
  end;
  WITH FileName DO
  begin
    Name := '';
    ErrorMessage := '';
  end;
  WITH FileName2 DO
  begin
    Name         := '';
    NumberPoints := '';
    ErrorMessage := '';
  end;
end;
```

{ This procedure saves the parameter values selected for use as defaults for the next run. }

```
procedure WriteDefault;

  VAR
    textfile : TEXT;

  begin {WriteDefault}
  Assign(textfile,'Default.dat');
  Rewrite(textfile);
  With Transducer do
  begin
    Writeln(textfile,Diameter);
    Writeln(textfile,RadiusOfC);
    Writeln(textfile,frequency);
    Writeln(textfile,pressure);
  end;
  With RunInput Do
  begin
    Writeln(textfile,MINM);
    Writeln(textfile,MAXM);
    Writeln(textfile,Npoints);
    Writeln(textfile,NDpoints);
    Writeln(textfile,DeltaR);
    Writeln(textfile,Zmin);
    Writeln(textfile,Zmax);
    Writeln(textfile,Rgmax);
    Writeln(textfile,RadiusMax);
  end;
  For n := 0 to 50 do
  WITH ZU[n] Do
  begin
    Writeln(textfile,location);
  end;
  Writeln(textfile,NumberBoundaryPoints);
  FOR n := 0 TO NumberBoundaryPoints - 1 DO
  begin
    VAL(BoundaryChoiceValues[n].location,temp1,code);
    VAL(BoundaryChoiceValues[n].NormalizedP,temp2,code);
    Writeln(textfile,Temp1:12:8);
    Writeln(textfile,Temp2:12:8);
  end;
  Close(textfile);
end;  {WriteDefault}
```

{ This procedure reads the parameter values selected for use as defaults for the next run. }

```
procedure ReadDefault;

  VAR
    textfile : TEXT;

  begin {ReadDefault}
  Assign(textfile,'Default.dat');
  Reset(textfile);
  With Transducer do
  begin
    Readln(textfile,Diameter);
    Readln(textfile,RadiusOfC);
    Readln(textfile,frequency);
    Readln(textfile,pressure);
    errormessage := '';
  end;
  With RunInput Do
  begin
    Readln(textfile,MINM);
    Readln(textfile,MAXM);
    Readln(textfile,Npoints);
    Readln(textfile,NDpoints);
    Readln(textfile,DeltaR);
    Readln(textfile,Zmin);
    Readln(textfile,Zmax);
    Readln(textfile,Rgmax);
    Readln(textfile,RadiusMax);
    errormessage := '';
  end;
  For n := 0 to 50 do
  WITH ZU[n] Do
  begin
    Readln(textfile,location);
    errormessage := '';
  end;
  READLN(textfile,NumberBoundaryPoints);
  FOR n := 0 TO NumberBoundaryPoints - 1 DO
  begin
    READLN(textfile,BoundaryChoiceValues[n].location);
    READLN(textfile,BoundaryChoiceValues[n].NormalizedP);
    BoundaryChoiceValues[n].errormessage := '';
  end;
  FOR n := NumberBoundaryPoints TO 100 DO
    WITH  BoundaryChoiceValues[n] DO
    begin
      location     := '';
      NormalizedP  := '';
      errormessage := '';
    end;
  Close(textfile);
  end;  {ReadDefault}
```

{ This procedure writes the input data file for the LFOCUS run. }

```
procedure WriteInputData;

  VAR
    textfile      : TEXT;
    tempstring    : STRING;
    tempval,
    radiusmaxval,
    deltarOut,
    radiusval     : DOUBLE;
    tempint       : INTEGER;

  begin {WriteInputData}
  Assign(textfile,'Input.dat');
  Rewrite(textfile);
  Writeln(textfile,'1');
  Writeln(textfile,'1.0d-6');
  Writeln(textfile,RunInput.RadiusMax);
  Writeln(textfile,RunInput.Rgmax+'DO');
  Writeln(textfile,'2.5d-14');
  Writeln(textfile,Transducer.pressure+'D+3');
  Writeln(textfile,'9.98d+2');
  Writeln(textfile,'5.5DO');
  Writeln(textfile,Transducer.RadiusOfC+'DO');
  VAL(Transducer.Diameter,tempval,code);
  tempval := tempval/2.0;
  STR(tempval:9:6,tempstring);
  Writeln(textfile,tempstring+'DO');
  Writeln(textfile,RunInput.DeltaR+'DO');
  Writeln(textfile,RunInput.Zmin+'DO');
  Writeln(textfile,RunInput.Zmax+'DO');
  Writeln(textfile,RunInput.Npoints);
  Writeln(textfile,Runinput.NDpoints);
  VAL(RunInput.NDpoints,Tempint,code);
  For n := 1 to tempint Do
  WITH ZU[n] Do
  begin
    Writeln(textfile,location+'DO');
  end;
  With RunInput Do
  begin
    Writeln(textfile,MINM);
    Writeln(textfile,MAXM);
  end;
  Writeln(textfile,'1.492D+3');
  Writeln(textfile,Transducer.frequency+'D+6');
  Writeln(textfile,NumberBoundaryPoints);
  FOR n := 0 TO NumberBoundaryPoints - 1 DO
  begin
    VAL(BoundaryChoiceValues[n].location,temp1,code);
    VAL(BoundaryChoiceValues[n].NormalizedP,temp2,code);
    Writeln(textfile,Temp1:12:8,'  ',Temp2:12:8);
  end;
  Close(textfile);
  end;  {WriteInputData}
```

```
{       This procedure writes a file which contains the location, normalized pressure pair that is used to describe the initial
        amplitude distribution at the face of the transducer.                                                                    }

procedure WriteBoundaryFile;

 VAR
  textfile : TEXT;
  n       : INTEGER;
  Temp1,
  Temp2    : DOUBLE;

begin {WriteBoundaryFile}

  Assign(textfile,FileName2.name);
  Rewrite(textfile);
  Writeln(textfile,FileName2.NumberPoints);
  FOR n := 0 TO NumberBoundaryPoints - 1 DO
  begin
    VAL(BoundaryChoiceValues[n].location,temp1,code);
    VAL(BoundaryChoiceValues[n].NormalizedP,temp2,code);
    Writeln(textfile,Temp1:12:8);
    Writeln(textfile,Temp2:12:8);
  end;
  CLOSE(textfile);

end;  {WriteBoundaryFile}

begin {Menu}
  ZU[0].location := '0.0';
  ReadDefault;
  InitiateBoundary_FileName;
  errorflag := TRUE;
  MyApp.Init;
  MyApp.Run;
  MyApp.Done;
  WriteDefault;
  IF BoundaryChoice.RadioButtonDataA = 2 THEN
    If BoundaryChoice.RadioButtonDataB = 1 THEN WriteBoundaryFile;
  WriteInputData;
 end. {Menu}
```

APPENDIX C

LFOCUS.F

PROGRAM LISTING

```
*       Some additional terms used in the program
*
*           ZMIN          - Minimum Z value for output purposes
*           ZMAX          - Maximum Z value for output purposes
*           RGMAX         - Maximum grid dimension
*           Rgmin         - Minimum grid dimension
*           Rmax          - Maximum R value
*           THRESHOLD     - Amplitude threshold when spectral term is added
*           MINM          - Minimum number of spectral terms in a calcualtion
*           MAXM          - Maximum number of spectral terms in the calcualtion
*           NPOINTS       - Number of normal longitudinal output points
*           NDPOINTS      - Number of detailed longitudinal output points
*           Number_BP     - Number of transverse boundary points (including 0)
*           TPOINTS       - Total number of output points
*           IOUTMAX       - Maximum transverse index for output
*           MARRAYINDEX   - Maximum array index
*           IMAX          - Maximum transverse array index
*           IMAX1         - IMAX - 1
*           IMAX4         - IMAX - 2
*           IMAXMIN       - Starting transverse array index
*           IMAXMAX       - Maximum transverse array index
*           RESTART       - Flag for restart after a controlled exit
*           K             - Counter for on-axis output
*           L             - Counter for off-axis output
*           COUNTER       - Local counter for fixing output points
*           CASE          - The current analysis case
*           NSTEP         - Step number
*           Pi            - Pi
*           TWOPi         - 2.0 * Pi
*           P0            - Initial On-axis pressure (kPa)
*           Omega         - Angular fundamental frequency
*           BoverA        - Nonlinearlity parameter (B/A)
*           Bover2A       - 1 + BoverA/2
*           Number_RP     - Number of uniformely spaced radial points
*           NewDeltaR     - Radial step size in meters
*
      DOUBLE PRECISION K1, K2, K3, KN1, KN2, KN3, KN4
      DOUBLE PRECISION ZMIN, ZMAX, Threshold, Pi, TwoPi
      DOUBLE PRECISION P0, Omega, BoverA, Bover2A, Rmax, Rgmin, Rgmax
      DOUBLE PRECISION OldDeltaR, NewDeltaR
      INTEGER CASE, MAXM, MINM, RESTART, IOUTMAX, NPOINTS, NDPOINTS, Number_BP
      INTEGER TPOINTS, COUNTER, IMAX, I, NSTEP, K, L, M, IMAXMIN
      INTEGER IMAXMAX, IMAX4, IMAX1, Number_RP
      LOGICAL NEWTEST
*
*       This is the first set of COMMON blocks
*
      COMMON /ALL/ NEWTEST
      COMMON /CONSTANTS/ A, D, Freq, C0, TwoPi, Pi
      COMMON /KVALUE/ K1, K2, K3, KN1, KN2, KN3, KN4
*
*       This common is used for indicees
*
      COMMON /INDEXES/ IMAX, M, IMAX1
*
*       This common is used by NEXT and the main program
*
      COMMON /NEXTC/ DeltaR, DeltaZ, Rayleigh, Ld, AlphaD, Rmax
*
*       At this point, take the maximum and minimum Z values and the specified
*       detailed output points and determine the points to output data.
*
```

```
*
*       The logic is simple. First calculate the location of each normal output
*       point. Then make a loop and use the loop to populate the ZOUT array. As
*       the array is populated, check to see if a detailed output point has been
*       reached. If so, add it to the array
*
*       Before reading the input data write one as the restart flag so that the
*       restart option can be exercised, if desired.
*
      OPEN(5,FILE = 'RESTART')
      READ(5,*) Restart
      CLOSE(5)
*
*       Set the maximum array index
*
      MARRAYINDEX = 4000
*
*       If restarting execution, reset the current parameter values to the ones present
*       when the run was terminated.
*
      IF (RESTART.EQ.1) THEN
*
*       If restarting, read the relevant paramters.
*
        OPEN(5,FILE='TEMP1.DAT')
        READ(5,*) Case
        READ(5,*) Threshold
        READ(5,*) IoutMax
        READ(5,*) Rgmax
        READ(5,*) Rgmin
        READ(5,*) Rmax
        READ(5,*) Alpha0
        READ(5,*) Alpha
        READ(5,*) AlphaD
        READ(5,*) P0
        READ(5,*) Ro0
        READ(5,*) BoverA
        READ(5,*) Bover2A
        READ(5,*) D
        READ(5,*) A
        READ(5,*) DeltaR
        READ(5,*) DeltaZ
        READ(5,*) Zmin
        READ(5,*) Zmax
        READ(5,*) Z
        READ(5,*) K
        READ(5,*) L
        READ(5,*) NSTEP
        READ(5,*) IMAXMAX
        READ(5,*) IMAXMIN
        READ(5,*) IMAX
        READ(5,*) IMAX4
        READ(5,*) K1
        READ(5,*) K2
        READ(5,*) K3
        READ(5,*) KN1
        READ(5,*) KN2
        READ(5,*) KN3
        READ(5,*) KN4
        READ(5,*) Npoints
        READ(5,*) NDpoints
        READ(5,*) Number_BP
        READ(5,*) Tpoints
        DO 299 I = 1, NDpoints
          READ(5,*) ZU(I)
299     CONTINUE
        DO 298 I = 0, Number_BP - 1
          READ(5,*) BoundaryP(0,I),BoundaryP(1,I)
298     CONTINUE
        READ(5,*) MinM
        READ(5,*) M
        READ(5,*) MaxM
        READ(5,*) C0
        READ(5,*) Freq
```

```
      READ(5,*) Omega
      READ(5,*) Rayleigh
      READ(5,*) Ld
      READ(5,*) Pi
      READ(5,*) TwoPi
*
*     Now, read the initially calcualted output array.
*
      DO 606 I = 1, Tpoints
        READ(5,*) ZOUT(I)
606   CONTINUE
      CLOSE(5)
*
*     Next, read the arrays used in the calculation G and H.
*
      CALL READTEMP(G, H, MAXM, IMAXMAX)
*
*     Now, open the old output files from the terminated run, set the pointer
*     to the end of the file, and return to the program.
*
334   OPEN(11,FILE='AMPZ',STATUS='OLD')
      READ(11,*,END=335) DUMMY
      GO TO 334
335   OPEN(12,FILE='PHASEZ',STATUS='OLD')
      READ(12,*,END=336) DUMMY
      GO TO 335
336   OPEN(13,FILE='AMPZR',STATUS='OLD')
      READ(13,*,END=337) DUMMY
      GO TO 336
337   OPEN(14,FILE='PHASEZR',STATUS='OLD')
      READ(14,*,END=338) DUMMY
      GO TO 337
338   BACKSPACE(11)
      BACKSPACE(12)
      BACKSPACE(13)
      BACKSPACE(14)
      DUMMY = DUMMY * 1.0
*
*     Jump to program execution.
*
      GOTO 333
     ENDIF
*
*     This is not a restart, therefore open the menu program.  But first set
*     restart to 1 for future reference.
*
     OPEN (5,FILE = 'RESTART')
     Restart = 1
     WRITE(5,397) Restart
397  FORMAT(I4)
     CLOSE(5)
*
*     Execute the menu program.
*
     CALL SYSTEM('Menu.exe')
*
*     This is the primary code input section.  For the integrated Pascal & FORTRAN
*     code, the file INPUT.DAT is the transfer point between the two codes.
*
     OPEN(5,FILE = 'INPUT.DAT')
     READ(5,*) Case
     READ(5,*) Threshold
     READ(5,*) IoutMax
     READ(5,*) Rgmax
     READ(5,*) Alpha0
     READ(5,*) P0
     READ(5,*) Ro0
     READ(5,*) BoverA
     READ(5,*) D
     READ(5,*) A
     READ(5,*) DeltaR
```

```
      READ(5,*) Zmin
      READ(5,*) Zmax
      READ(5,*) Npoints
      READ(5,*) NDpoints
      Do 399 I = 1, NDpoints
        READ(5,*) ZU(I)
399   CONTINUE
      READ(5,*) MinM
      READ(5,*) MaxM
      READ(5,*) C0
      READ(5,*) Freq
    READ(5,*) Number_BP
    Do 297 I = 0, Number_BP - 1
      READ(5,*) BoundaryP(0,I),BoundaryP(1,I)
297 CONTINUE
*
*       Close the file and calculate some of the parameters
*
      CLOSE(5)
*
*       Set the boundary region Rgmin to 10% greater than the physical edge
*       of the transducer.
*
      Rgmin     = 1.1D0
*
*       Calculate B/2A using B/A
*
      Bover2A   = 1.0D0 + (BoverA/2.0D0)
*
*       Set the values of Pi and 2 Pi for the program
*
      Pi        = 2.0D0 * DASIN(1.0D0)
      TwoPi     = 2.0D0 * Pi
*
*       Set Omega and the attenuation terms
*
      Omega     = TwoPi * Freq
      Alpha     = Alpha0 * (Freq**2)
      AlphaD    = Alpha * D
*
*       Set the total number of points
*
      TPOINTS = NPOINTS + NDPOINTS
*
*       Now open the output files
*
      OPEN(11,File = 'AMPZ')
      OPEN(12,File = 'PHASEZ')
      OPEN(13,FILE = 'AMPZR')
      OPEN(14,FILE = 'PHASEZR')
*
*       To identify the output later include the input data a file header.
*
      DO 237 I = 11,14
        CALL HEADER(I,Case,D,A,P0,DeltaR,Zmin,Zmax,IoutMax,Npoints,NDpoints,Number_BP,Tpoints,
     +              ZU,MinM,MaxM,Freq,BoundaryP)
237   CONTINUE
*
*       Set Z to ZMIN, COUNTER to 1, and populate the output point array.
*
      DUMMY = (ZMAX - ZMIN)/NPOINTS
      Z = ZMIN
      COUNTER = 1
      DO 10 I = 1, TPOINTS
        Z = Z + DUMMY
        IF ((Z.GE.ZU(COUNTER)).AND.(COUNTER.LE.NDPOINTS)) THEN
          Z = Z - DUMMY
          ZOUT(I) = ZU(COUNTER)
          COUNTER = COUNTER + 1
        ELSE
```

```
        ZOUT(I) = Z
      ENDIF
10    CONTINUE
*
*     Now initialize a number of program parameters.
*
*     First, calcualte the K values used in the differencing scheme.
*
      K1  = -1.0D0/12.0D0
      K2  = 16.0D0/12.0D0
      K3  = -30.0D0/12.0D0
      KN1 = 1.0D0/90.0D0
      KN2 = -1.0D0 * ((1.0D0/12.0D0) + (2.0D0/30.0D0))
      KN3 = (16.0D0/12.0D0) + (15.0D0/90.0D0)
      KN4 = -1.0D0 * ((30.0D0/12.0D0) + (2.0D0/9.0D0))
*
*     Then the rest
*
      IMAXMAX  = INT((Rgmax/DeltaR) + 1.0D-5)
*
*     This keeps the data entry errors from blowing up the code.
*
      IF (IMAXMAX.GT.MARRAYINDEX) THEN
        IMAXMAX = MARRAYINDEX
      ENDIF
*
*     Continue with the calculation.
*
      IMAXMIN   = INT((Rgmin/DeltaR) + 1.0d-5)
      IMAX      = IMAXMIN
      IMAX4     = IMAX - 4
      K         = 1
      L         = 1
      NSTEP     = 0
      Rayleigh = Omega * A * A/(2.0D0 * C0)
*
*     For later use save the initial value of DeltaR.
*
      OldDeltaR = DeltaR
*
      DeltaR    = DeltaR * DSQRT(Rayleigh/D)
      Rmax      = DSQRT(Rayleigh/D)
*
*     Use 0.4 in the DeltaZ conversion process to make the calculations more stable.
*
      DeltaZ    = 0.4d0 * (DeltaR ** 2)
      Z         = -DeltaZ
*
*     Calculate the initial discontinuity distance.
*
      Ld        = Ro0 * (C0**3)/(Bover2A * Omega * P0)
      M         = Minm
*
*     Now run the initialization programs.  Remember the initialization has to
*     be at the maximum grid points.  First set the transverse boundary condition.
*
    CALL SET_BOUNDARY(A, Number_BP, Radial_P, BoundaryP, NewDeltaR, Number_RP, OldDeltaR, IMAXMAX)
*

      CALL INITIAL(G, H, IMAXMAX, MAXM, DeltaR)
*
*     This is the rentry point for a restart.
*
333   CONTINUE
*
      IMAX1 = IMAX - 1
*
*     To allow program restart, set the control break option.  Once Ctrl C is
```

```
*        pressed, the program will jump to the designated point, once detected.
*
      NEWTEST = .FALSE.
      OPTION BREAK(NEWTEST)
*
*        Set the major program loop
*
100   NSTEP = NSTEP + 1
      Z = Z + DeltaZ
      IF (Z.GT.ZOUT(K)) THEN
        Z = ZOUT(K)
      ENDIF
      CALL NEXT(G,H)
*
*        This simple four line section of code floats the transverse boundary.
*        Calculate the amplitude four cells from the rigid boundary. If it has
*        exceeded the threshold, expand if necessary. This works because the lowest
*        index value corresponds to the lowest frequency.
*
      IF (IMAX.LT.IMAXMAX) THEN
        IF (Ampltu(G(IMAX4,1),H(IMAX4,1)).GT.Threshold) THEN
          IMAX = IMAX + 2
          IF (IMAX.GE.IMAXMAX) IMAX = IMAXMAX
          IMAX1 = IMAX - 1
          IMAX4 = IMAX - 4
        ENDIF
      ENDIF
*
*        Now see whether the on-axis output is necesssary by checking to see if the
*        value corresponds to an element of zout(k).
*
      IF (Z.EQ.ZOUT(K)) THEN
*
*        At each output point, write the on-axis amplitude and phase spectra.
*
        WRITE(11,1009) M , Z, (AMPLTU(G(0,N),H(0,N)),N = 0,M)
        WRITE(12,1009) M , Z, (PHASE(G(0,N),H(0,N)),N = 0,M)
        WRITE(*,500) Nstep, z, k, m, Imax
500     FORMAT(' Nstep = ',I6,' z = ',F10.6,' k = ',I3,' m = ',I3,' Imax = ',I4)
        K = K + 1
      ENDIF
1009  FORMAT(1P,I4,' ',500E16.4E4)
      IF(Z.EQ.ZU(L)) THEN
*
*        If z is at the pont z(l) the increment the index in the zu(l) array.
*
        IF(L.LT.NDPOINTS) THEN
          L = L + 1
        ELSE
          L = NDPOINTS
        ENDIF
*
*        This is the place to output the spectra to ampr and phaser.
*
        DO 2000 INEW = 0, IOUTMAX
          WRITE(13,1009) M, Z, (OldDeltaR * INEW), (AMPLTU(G(INEW,N),H(INEW,N)),N=0,M)
          WRITE(14,1009) M, Z, (OldDeltaR * INEW), (PHASE(G(INEW,N),H(INEW,N)),N=0,M)
2000    CONTINUE
      ENDIF
      IF (M.LT.MAXM) THEN
        IF(AMPLTU(G(0,M),H(0,M)).GT.Threshold) THEN
          M = M + 1
        ENDIF
      ENDIF
      IF(K.EQ.(TPOINTS+1)) GO TO 115
*
*        If ^C has been pressed the go to 905.
*
      IF(NEWTEST) GOTO 905
      GOTO 100
115   CONTINUE
```

```
*
*     If finished, then reset restart to 0.
*
      OPEN(5,FILE='RESTART')
      RESTART = 0
      WRITE(5,*) RESTART
      CLOSE(5)
*
*     Now jump to 906.
*
      GOTO 906
905   CONTINUE
*
*     If restarting, then write the relevant paramters.
*
      OPEN(5,FILE='TEMP1.DAT')
      WRITE(5,*) Case
      WRITE(5,*) Threshold
      WRITE(5,*) IoutMax
      WRITE(5,*) Rgmax
      WRITE(5,*) Rgmin
      WRITE(5,*) Rmax
      WRITE(5,*) Alpha0
      WRITE(5,*) Alpha
      WRITE(5,*) AlphaD
      WRITE(5,*) P0
      WRITE(5,*) Ro0
      WRITE(5,*) BoverA
      WRITE(5,*) Bover2A
      WRITE(5,*) D
      WRITE(5,*) A
      WRITE(5,*) DeltaR
      WRITE(5,*) DeltaZ
      WRITE(5,*) Zmin
      WRITE(5,*) Zmax
      WRITE(5,*) Z
      WRITE(5,*) K
      WRITE(5,*) L
      WRITE(5,*) NSTEP
      WRITE(5,*) IMAXMAX
      WRITE(5,*) IMAXMIN
      WRITE(5,*) IMAX
      WRITE(5,*) IMAX4
      WRITE(5,*) K1
      WRITE(5,*) K2
      WRITE(5,*) K3
      WRITE(5,*) KN1
      WRITE(5,*) KN2
      WRITE(5,*) KN3
      WRITE(5,*) KN4
      WRITE(5,*) Npoints
      WRITE(5,*) NDpoints
      WRITE(5,*) Number_BP
      WRITE(5,*) Tpoints
      DO 199 I = 1, NDpoints
        WRITE(5,*) ZU(I)
199   CONTINUE
      DO 296 I = 0, Number_BP - 1
        WRITE(5,*) BoundaryP(0,I),' ',BoundaryP(1,I)
296   CONTINUE
      WRITE(5,*) MinM
      WRITE(5,*) M
      WRITE(5,*) MaxM
      WRITE(5,*) C0
      WRITE(5,*) Freq
      WRITE(5,*) Omega
      WRITE(5,*) Rayleigh
      WRITE(5,*) Ld
      WRITE(5,*) Pi
      WRITE(5,*) TwoPi
```

```
*
*     Now write the output array that was initially calculated.
*
      DO 406 I = 1, Tpoints
        WRITE(5,*) ZOUT(I)
406   CONTINUE
      CLOSE(5)
*
*     Next write the arrays used for G and H in the calculation.
*
      CALL WRITETEMP(G, H, MAXM, IMAXMAX)
*
906   CONTINUE
      WRITE(*,*) 'PROGRAM EXECUTION COMPLETED'
*
*  Now close the output files
*
      CLOSE(11)
      CLOSE(12)
      CLOSE(13)
      CLOSE(14)
      STOP
      END
```

```
      SUBROUTINE HEADER(N,Case,D,A,PO,DeltaR,Zmin,Zmax,IoutMax,Npoints,NDpoints,Number_BP,Tpoints,
     +                  ZU,MinM,MaxM,Freq,BoundaryP)
*
*      This subroutine writes a header on each output file
*
      DOUBLE PRECISION Zmin, Zmax, DeltaR, D, A, Freq, ZU(50), PO, BoundaryP(0:1,0:100)
      INTEGER CASE, I, Npoints, NDpoints, Tpoints, MinM, MaxM, N, IoutMax, Number_BP
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
*
*    This is the collection of write statements
*
      WRITE(N,*) CASE
      WRITE(N,1000) D
      WRITE(N,1000) A
      WRITE(N,1000) PO
      WRITE(N,1000) DeltaR
      WRITE(N,1000) Freq
      WRITE(N,1000) Zmin
      WRITE(N,1000) Zmax
      WRITE(N,*) IoutMax
      WRITE(N,*) Npoints
      WRITE(N,*) NDpoints
      WRITE(N,*) Tpoints
      WRITE(N,*) MinM
      WRITE(N,*) MaxM
      DO 1002 I = 1,NDpoints
        WRITE(N,1000) ZU(I)
1002  CONTINUE
      DO 1003 I = 0, Number_BP - 1
        WRITE(N,*) BoundaryP(0,I),' ',BoundaryP(1,I)
1003  CONTINUE
1000  FORMAT(1P,E15.6E3)
      RETURN
      END
```

```
      SUBROUTINE READTEMP(G,H,MAXM,IMAX)
      DOUBLE PRECISION G(0:4000,0:30), H(0:4000,0:30)
      INTEGER MAXM,IMAX
*
*       This subroutine reads files Temp2.dat through Temp5.dat.
*
      OPEN(5,FILE='TEMP2.DAT')
      DO 607 I=0,MAXM/2
        DO 608 J = 0,IMAX
          READ(5,*) G(J,I)
608     CONTINUE
607   CONTINUE
      CLOSE(5)
      OPEN(5,FILE='TEMP3.DAT')
      DO 307 I=(MAXM/2)+1,MAXM
        DO 308 J = 0,IMAX
          READ(5,*) G(J,I)
308     CONTINUE
307   CONTINUE
      CLOSE(5)
      OPEN(5,FILE='TEMP4.DAT')
      DO 617 I=0,MAXM/2
        DO 618 J = 0,IMAX
          READ(5,*) H(J,I)
618     CONTINUE
617   CONTINUE
      CLOSE(5)
      OPEN(5,FILE='TEMP5.DAT')
      DO 317 I=(MAXM/2)+1,MAXM
        DO 318 J = 0,IMAX
          READ(5,*) H(J,I)
318     CONTINUE
317   CONTINUE
      CLOSE(5)
      RETURN
      END
```

```
      SUBROUTINE WRITETEMP(G,H,MAXM,IMAX)
      DOUBLE PRECISION G(0:4000,0:30), H(0:4000,0:30)
      INTEGER MAXM,IMAX
*
*     This subroutine writes Temp2.dat through Temp5.dat
*
 50   FORMAT(1P,4D30.15)
      OPEN(5,FILE='TEMP2.DAT')
      DO 100 I=0,MAXM/2
        DO 90 J = 0,IMAX
          WRITE(5,*) G(J,I)
 90     CONTINUE
 100  CONTINUE
      CLOSE(5)
      OPEN(5,FILE='TEMP3.DAT')
      DO 200 I=(MAXM/2)+1,MAXM
        DO 190 J = 0,IMAX
          WRITE(5,*) G(J,I)
 190    CONTINUE
 200  CONTINUE
      CLOSE(5)
      OPEN(5,FILE='TEMP4.DAT')
      DO 300 I=0,MAXM/2
        DO 290 J = 0,IMAX
          WRITE(5,*) H(J,I)
 290    CONTINUE
 300  CONTINUE
      CLOSE(5)
      OPEN(5,FILE='TEMP5.DAT')
      DO 400 I=(MAXM/2)+1,MAXM
        DO 390 J = 0,IMAX
          WRITE(5,*) H(J,I)
 390    CONTINUE
 400  CONTINUE
      CLOSE(5)
      RETURN
      END
```

```
      SUBROUTINE SET_BOUNDARY(A, Number_BP, Radial_P, BoundaryP, NewDeltaR, Number_RP,
                              OldDeltaR, Imax)
*
*     This subroutine calculates the initial normalized pressure distribution
*
*     Declared variables
*
*     A              = Transducer radius
*     Number_BP      = Number of experimental values
*     Number_RP      = Number of uniformly spaced radial points
*     BoundaryP      = Experimental point 2-D array
*     OldDeltaR      = Normalized step size
*     NewDeltaR      = Radial step size (meters)
*     Radius         = Radial location (meters)
*     Radial_P       = Uniformly spaced normalized pressure values
*     LocalIndex     = Array index for incrementing BoundaryP
*     I              = Loop index
*     Imax           = Maximum number of transverse terms
*     m              = Slope
*     b              = Y intercept
*
      INTEGER Number_BP, LocalIndex, I, Number_RP, Imax
      DOUBLE PRECISION Radial_P(0:4000), BoundaryP(0:1,0:100), NewDeltaR
      DOUBLE PRECISION OldDeltaR, Radius, m, b, a
*
*     Set number of radial points. This value is passed to main program.
*
      Number_RP = IDINT(1.0d0/OldDeltaR)
      IF (DMOD(1.0d0,OldDeltaR).NE.0.0d0) Number_RP = Number_RP + 1
*
*     Given the number of radial points, set the pressure values. For a
*     uniform distribution set the pressure to 1 and go to the end.
*
      IF (Number_BP.EQ.1) THEN
        DO 100 I = 0, Number_RP
          Radial_P(I) = 1.0d0
100     CONTINUE
        DO 110 I = Number_RP +1, Imax
          Radial_P(I) = 0.0d0
110     CONTINUE
        GOTO 500
      ENDIF
*
*     Now if we don't have a uniform distribution, construct a profile
*     based on the experimental points. Set the on axis point (0),
*     the local index, and radial step size (in meters).
*
      Radial_P(0) = BoundaryP(1,0)
      LocalIndex = 1
      NewDeltaR = OldDeltaR * A
*
*     Now build the array
*
      DO 200 I = 1, Number_RP
        Radius = I * NewDeltaR
*
*     Check whether the radius is < BoundaryP(0,localindex). If so,
*     interpolate to determine Radial_P(I).
*
210     CONTINUE
        IF (Radius.LE.BoundaryP(0,LocalIndex)) THEN
          m = (BoundaryP(1,LocalIndex) - BoundaryP(1,LocalIndex - 1))
          m = m/(BoundaryP(0,LocalIndex) - BoundaryP(0,LocalIndex - 1))
          b = BoundaryP(1,LocalIndex) - (m * BoundaryP(0,LocalIndex))
          Radial_P(I) = (m * Radius) + b
          ELSE
*
*     If not, increment the index and loop back.
```

```
*
            IF (LocalIndex .LT. (Number_BP - 1)) THEN
              LocalIndex = LocalIndex + 1
              GOTO 210
              ELSE
              Radial_P(I) = BoundaryP(1,LocalIndex)
              ENDIF
            ENDIF
 200      CONTINUE
          DO 220 I = Number_RP +1, Imax
            Radial_P(I) = 0.0d0
 220      CONTINUE
 500      CONTINUE
*
*         Now write the results to check the process.
*
          OPEN(5,File = 'TempFile.dat')
          Write(5,*) Number_RP
          Write(5,*) NewDeltaR
          Write(5,*) A
          Write(5,*) Number_BP
          Write(5,*) OldDeltaR
          Do 300 I = 0, Imax
            RADIUS = I * NewDeltaR
            Write(5,*) Radius,' ',Radial_P(I)
 300      CONTINUE
          CLOSE(5)
*
*         Stop and return.
*
          RETURN
          END
```

```
      SUBROUTINE INITIAL(G, H, IMAX, MAXM, DeltaR)
*
*       This subroutine Initializes the coefficients G and H at the transverse boundary.
*
      DOUBLE PRECISION G(0:4000,0:30), H(0:4000,0:30), U, DeltaR
      DOUBLE PRECISION INIAMP, INIPHS, A, D, Freq, CO, TwoPi, Pi
      INTEGER MAXM, IMAX, I, N
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
      COMMON /CONSTANTS/ A, D, Freq, CO, TwoPi, Pi
*
*       Since only a single harmonic exists at the start, initalize G and H of (i,1).
*
      DO 10 I=0,IMAX
        U = I * DeltaR
        IF (Radial_P(I).NE.0.0D0) THEN
          G(I,1) = Radial_P(I)*DCOS(INIPHS(U))
          H(I,1) = Radial_P(I)*DSIN(INIPHS(U))
        ELSE
          G(I,1) = 0.0D0
          H(I,1) = 0.0D0
        ENDIF
10    CONTINUE
*
*       For safety's sake set all other terms to 0
*
      DO 20 I = 0, IMAX
        DO 30 N = 2, MAXM
          G(I,N) = 0.0D0
          H(I,N) = 0.0D0
30      CONTINUE
20    CONTINUE
      RETURN
      END
```

```
      DOUBLE PRECISION FUNCTION INIPHS(U)
*
*      This procedure adjusts the phase of the input pressure front so that it
*      appears to be comming from a curved surface.
*
      DOUBLE PRECISION A, D, Rvalue, U, Freq, CO, TwoPi, Pi, K
      DOUBLE PRECISION DeltaR, DeltaZ, Rayleigh, Ld, ALphaD, Rmax
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
      COMMON /CONSTANTS/ A, D, Freq, CO, TwoPi, Pi
      COMMON /NEXTC/ DeltaR, DeltaZ, Rayleigh, Ld, AlphaD, Rmax
*
*      Convert from normalized radial coordinates to real coordinates.
*
      Rvalue = A * U * DSQRT(D/(4.0D0 * Rayleigh))
*
*  Calculate the wave number based on the input frequency
*
      K = TwoPi * Freq/CO
*
*     Phase Factor
*
      INIPHS = K * (DSQRT(D*D + Rvalue*Rvalue) - D)
*
      RETURN
      END
```

```
      SUBROUTINE NEXT(G,H)
*
*       This subroutine calculates G and H for the next longitudinal distance step.
*
      DOUBLE PRECISION G(0:4000,0:30), H(0:4000,0:30)
      DOUBLE PRECISION GNEW(0:4000), HNEW(0:4000), NORM
      DOUBLE PRECISION GFN(0:4000), HFN(0:4000), GOLD(0:4000), Rmax
      DOUBLE PRECISION SUMG, SUMH, Eps, Rayleigh, A, D, Freq, CO, C3
      DOUBLE PRECISION C1, C2, DeltaR, DeltaZ, Ld, AlphaD, TwoPi, Pi
      INTEGER M, IMAX, LMAX, IMAX1
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
      COMMON /CONSTANTS/ A, D, Freq, CO, TwoPi, Pi
      COMMON /NEXTC/ DeltaR, DeltaZ, Rayleigh, Ld, AlphaD, Rmax
      COMMON /INDEXES/ IMAX, M, IMAX1
*
*       Set the local convergence constants
*
      Eps = 1.0d-5
      LMAX = 20
*
*       These constants are calculated outside the loop to speed up the program.
*
      DO 60 N=1,M
        C1 = DeltaZ * N * D/(2.0D0 * Ld)
        C2 = 1.0D0 /(1.0D0 + (DeltaZ * AlphaD * N * N))
        C3 = DeltaZ/(4.0D0 * N * DeltaR * DeltaR)
*
*       This loop calculates the non-linear term contribution.
*
        DO 10 I=0,IMAX1
          GOLD(I) = G(I,N)
          GFN(I) = GOLD(I)+C1*SUMG(G,H,N,M,I)
          HNEW(I) = H(I,N)
          HFN(I) = HNEW(I)+C1*SUMH(G,H,N,M,I)
10      CONTINUE
*
*       L is the convergence counter.  If L exceeds 20, the program aborts.
*
        L = 1
*
*       Apply the seven point Lapacian
*
20      CALL CONVOL(GNEW,GFN,C2,C3,HNEW)
        CALL CONVOL(HNEW,HFN,C2,-C3,GNEW)
*
*       If the step to step error exceeds the convergence factor, continue to iterate.
*
        DO 50 I=0,IMAX1
          NORM = DABS(GNEW(I) - GOLD(I))
          IVAL = I
          IF (NORM.GT.EPS) GOTO 77
50      CONTINUE
*
*       If the setp to step error is allowable, jump to the next step.
*
        GO TO 54
77      CONTINUE
        DO 59 I=0,IMAX1
          GOLD(I) = GNEW(I)
59      CONTINUE
*
*       If L ≥ 20, then terminate the program and write the diagnostics.
*
        IF (L.GT.LMAX) THEN
          WRITE(*,*) 'ITERATION FOR N = ',N,' FAILED. PROGRAM ABORT'
          WRITE(*,885) NORM, Eps, L, IVAL
885       FORMAT(1p,2d15.6,2I5)
          STOP
        ENDIF
```

```
*
*     Increment L and continue to iterate
*
      L = L+1
      GO TO 20
*
*     Set G and H to the new values
*
54    DO 53 I  = 0,IMAX1
        G(I,N) = GNEW(I)
        H(I,N) = HNEW(I)
53    CONTINUE
55    CONTINUE
60    CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CONVOL(Z,X,C2,C3,Y)
*
*       This subroutine performs the seven point laplacian operator at each point in the Array Y.
*
      DOUBLE PRECISION Z(0:4000), Y(0:4000), X(0:4000), C2, C3, K1, K2, K3
      DOUBLE PRECISION KN1, KN2, KN3, KN4
      INTEGER IMAX, M, IMAX1
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
      COMMON /INDEXES/ IMAX, M, IMAX1
      COMMON /KVALUE/ K1, K2, K3, KN1, KN2, KN3, KN4
*
*       Z        = The output value which is G or H
*       X        = The previous girdpoint value which is G or H
*       Y        = The current grid value which is H or G
*       C2       = 1.0D0 /(1.0D0 + (DeltaZ * AlphaD * N * N))
*       C3       = D * DeltaZ/(4.0D0 * Z0 * N * DeltaR * DeltaR)
*
*       Calcualte the on-axis term, and the next two terms moving away from the axis
*
      Z(0) = C2 *(X(0)+ C3 * (KN1 * (Y(3) + Y(3)) + KN2 * (Y(2) + Y(2)) + KN3 * (Y(1) + Y(1))
     +            + KN4 * Y(0)))
      Z(1) = C2 *(X(1)+ C3 * (KN1 * (Y(4) + Y(2)) + KN2 * (Y(3) + Y(1)) + KN3 * (Y(2) + Y(0))
     +            + KN4 * Y(1)))
      Z(2) = C2 *(X(2)+ C3 * (KN1 * (Y(5) + Y(1)) + KN2 * (Y(4) + Y(0)) + KN3 * (Y(3) + Y(1))
     +            + KN4 * Y(2)))
*
*       Next compute the terms up to the IMAX - 4
*
      DO 10 I = 3, IMAX - 4
        Z(I) = C2 *(X(I)+ C3 * (KN1 * (Y(I+3) + Y(I-3)) + KN2 * (Y(I+2) + Y(I-2))
     +            + KN3 * (Y(I+1) + Y(I-1)) + KN4 * Y(I)))
10    CONTINUE
*
*       End by calcualting the terms at the upper boundary
*
      Z(IMAX-3) = C2 * (X(IMAX-3)+  C3 * (K1 * (Y(IMAX-1) + Y(IMAX-5)) + K2 * (Y(IMAX-2)
     +            + Y(IMAX-4)) + K3 * Y(IMAX-3)))
      Z(IMAX-2) = C2 * (X(IMAX-2) + C3 * (Y(IMAX-1) - 2.0D0 * Y(IMAX-2) + Y(IMAX-3)))
      Z(IMAX-1) = C2 * (X(IMAX-1) + C3 * (Y(IMAX-2) - 2.0D0 * Y(IMAX-1)))
*
*       Return to the main program
*
      RETURN
      END
```

```
*
*     This pair of functions is used to calculate linearized sum term at
*     each longitudinal step
*

      DOUBLE PRECISION FUNCTION SUMG(G,H,N,M,I)
*
*     This function calcualtes the linearized contriubtion for G
*
      DOUBLE PRECISION G(0:4000,0:30), H(0:4000,0:30), SUM
      INTEGER K, N, M, I
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
      SUM = 0.0D0
      DO 10 K =1,(N-1)/2
        SUM = SUM+G(I,N-K)*G(I,K)-H(I,N-K)*H(I,K)
10    CONTINUE
      IF (MOD(N,2).EQ.0) THEN
        SUM = SUM + (0.5*(G(I,N/2)**2 - H(I,N/2)**2))
      ENDIF
      DO 20 K=M,(N+1),-1
        SUM = SUM-G(I,K-N)*G(I,K)-H(I,K-N)*H(I,K)
20    CONTINUE
*
      SUMG = SUM
*
      RETURN
      END
*
*
*

      DOUBLE PRECISION FUNCTION SUMH(G,H,N,M,I)
*
*     This function calcualtes the linearized contribution for H.
*
      DOUBLE PRECISION G(0:4000,0:30), H(0:4000,0:30), SUM
      INTEGER K, N, M, I
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
      SUM = 0.0D0
      DO 10 K=1,(N-1)
        SUM = SUM+G(I,N-K)*H(I,K)
10    CONTINUE
      DO 20 K=M,(N+1),-1
        SUM = SUM+G(I,K)*H(I,K-N)-H(I,K)*G(I,K-N)
20    CONTINUE
*
      SUMH = SUM
*
      RETURN
      END
```

```
*
*     The following functions are used to calculate the amlitude and
*     phase of the spectral terms given the local values of G and H.
*
      DOUBLE PRECISION FUNCTION AMPLTU(G,H)
*
*     This function calcualtes the normalized presure amplitude of the nth harmonic.
*
      DOUBLE PRECISION G, H
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
      AMPLTU = DSQRT(G*G+H*H)
      RETURN
      END
*
*
*
      DOUBLE PRECISION FUNCTION PHASE(G,H)
*
*     This function returns the nth harmonic's phase in degrees.
*
      DOUBLE PRECISION G, H, MAG
      LOGICAL NEWTEST
      COMMON /ALL/ NEWTEST
*
*     Calculate the magnitude of the spectral term
*
      MAG = DSQRT(G*G+H*H)
      IF (MAG.EQ.0.) THEN
        PHASE = 0.0D0
      ELSE
        PHASE = DATAN2(H,G)*90.0D0/DASIN(1.0D0)
      ENDIF
*
*     Make sure the phase is a number between 0 and 360 degrees.
*
      IF (PHASE.GE.-180.D0) GO TO 20
10    PHASE = PHASE+360.0D0
      IF (PHASE.LE.-180.D0) GO TO 10
20    RETURN
      END
```

Phase of Lamb wave radiation from a plate immersed in a liquid

J.-F. Piet, G. Quentin, M. de Billy, Q.-G. Zhu† and W.G. Mayer†

Equipe d'Optique Ultrasonore, GPS, Université Paris 7, 75251 Paris Cedex 05, France
† Ultrasonics Research Laboratory, Physics Department Georgetown University, Washington, DC 20057, USA

Received 28 February 1992

A stroboscopic Schlieren image of ultrasonic radiation from a plate immersed in water is presented and is compared with the calculated radiation pattern for the plate in which a leaky Lamb mode has been generated. The phase relationship of the radiation lobes created at the top, bottom, and narrow end surfaces of the plate are discussed.

Keywords: leaky Lamb modes; Schlieren imaging; radiation pattern phase

It was recently shown[1,2] that Lamb waves in a solid plate immersed in water radiate some of their energy from the end of the plate. This radiation from the end of the plate consists of an odd number of lobes when the plate carries a symmetric Lamb mode and an even number of lobes when the mode is antisymmetric. The previously published Schlieren images produced by continuous waves and theoretical calculations of the relative intensity distribution between the lobes[2], as well as hydrophone measurements[1] do not indicate the respective phase of the radiation. Moreover, the use of continuous waves incident from the water onto the plate to generate a given leaky Lamb mode also produces specular and non-specular reflections, first observed by Schoch[3], and later explained and described by Neubauer[4], Bertoni and Tamir[5], and others[6,7]. When these reflections of the incident continuous wave occur near the end of the plate, they may interfere with other possible reradiations from the end of the plate. The Schlieren images published by Zhu *et al.*[1] clearly show reflections from, and transmissions through, the plate which are caused by such interferences. They are not specular or non-specular reflections since the radiation is not taking place at the expected Lamb angle.

However, the numerical approach used to find the intensity distribution of the radiation at the end of the plate[2] does contain information from which relative phases of the waves in the lobes can be determined. Furthermore, the numerical method can be used to describe radiation from the top and the bottom of the plate, near its end, which does not include specular and non-specular reflections. The latter are simply a consequence of a particular technique of exciting Lamb modes and, thus, may be considered not to be a part of the radiation from a Lamb mode *per se*.

This paper is concerned with the phase and the amplitude distribution of the radiation occurring at the end of the plate as well as from the top and the bottom near the plate's end. The interference by incident or reflected waves has been eliminated in both the mathematical model and the experimental arrangement described.

Calculational approach

The basis for the calculations are the finite different methods discussed by Bond[8] and Harker[9], adopted for the present case of energy transfer from the particle displacement at the end surfaces of an aluminium plate to the surrounding water. This particle displacement can be calculated for a plate vibrating in a Lamb mode which is excited by a longitudinal ultrasonic wave in water, incident at the appropriate Lamb angle.

Assuming that the long dimension of the plate is the x-direction, the thickness of the plate is in the z-direction and the Lamb wave propagates in the x-direction, then the wave motion in the homogeneous solid plate can be expressed as

$$(\lambda + \mu)\nabla\nabla \cdot u + \mu\nabla^2 u = \rho \frac{\partial^2 u}{\partial t^2} \tag{1}$$

Here λ and μ are the Lamé constants, ρ is the density of the plate material, and u is the displacement vector which has an x and y component and is time dependent. The y-direction can be omitted assuming the plate to be wide enough to eliminate any y-dependence. With the velocities of the longitudinal and shear waves given, respectively by

$$p^2 = (\lambda + \mu)/\rho \tag{2}$$

$$s^2 = \mu/\rho \tag{3}$$

The general equations for the particle displacements in

the x and z directions, u and v, respectively, are then

$$u_{tt} = p^2 u_{xx} + (p^2 - s^2) v_{xz} + s^2 u_{zz}, \quad (4)$$

$$v_{tt} = p^2 v_{zz} + (p^2 - s^2) u_{xz} + s^2 v_{xx}, \quad (5)$$

the subscripts indicate derivatives.

The Lamb wave to be examined is produced by allowing an ultrasonic beam of frequency f (with f between 2 and 10 MHz) to be incident at the appropriate angle from the water on to the plate so that the desired Lamb mode is set up in the plate. Other parameters needed for a numerical analysis are the beam profile, beam width, plate thickness d, and distance from the point of beam incidence on the plate to the end of the plate. In the present case the plate was 0.9 mm thick, and the distance beam to the end of plate could be varied. The beam profile was assumed to be Gaussian.

Equations (4) and (5) can then be solved by the FDM with incremental steps of x of about 0.1λ and the increments of t are selected so that

$$\Delta t \leqslant \Delta x(p^2 + s^2)^{-1/2} \quad (6)$$

If the size of the time and space increments are selected so that Equation (6) is satisfied, the accuracy and stability of the calculation will be assured[10].

Numerical calculations

The above method was used to calculate the radiation pattern at the end of the plate vibrating in the A_1-Lamb mode. The frequency of the continuous wave was assumed to be 3.29 MHz and the plate thickness 0.9 mm, resulting in an fd-value of 2.96 MHz mm. The A_1-Lamb mode will be excited for these parameters if the angle of incidence in water is 13.6°.

This set of parameters is used as the input. The output is an array of calculated sound-intensity values for 43 points along the x-direction and 34 points in the z-direction. Since calculations are made at incremental distances of 0.1 mm in these two directions, the mapped area is 4.2 mm by 3.3 mm, which means that there are 1462 calculated intensity values in the mapped area.

Gridding software is used to establish as many additional intensity values as desired, located between the calculated data points, and to create contour lines connecting identical intensity values. Software was also used to find the maximum value of the intensity in the radiated energy in the water in the mapped area which, in the case discussed here, was approximately 0.033 of the maximum value of the intensity of the Gaussian beam incident at the water–solid interface of the plate.

Figure 1 shows the intensity distribution in the mapped area. The conour lines are 1 dB apart, starting with those lines connecting the maximum intensity points in the lobes and ending with those lines connecting intensity points −10 dB below that value. No lines are drawn in areas where the intensity is more than 10 dB below the maximum. All points with intensity values above the maximum in water are connected with one line, regardless of how much higher the intensity value is at those points, compared with the maximum intensity in water. Clearly this results in a dark area, representing the plate where the intensity is generally very much higher than in the reradiation pattern in the lobes in the water.

The intensity representation in *Figure 1* depicts the situation at a given time increment, in this case 34 periods after the start of the generation of the Lamb wave in the plate. Therefore, the reradiation pattern around the end of the plate is well established, the high-intensity areas in the water represent the location of the positive and negative maximum pressure amplitudes in the lobes, i.e. the $\frac{1}{2}\lambda$ locations of the 3.29 MHz wave in the water. Based on this particular appearance of the low lobes radiating from the end of the plate one cannot determine whether the water in the lobes are in phase or 180° out of phase as they leave the plate. This is also the case for a Schlieren image of the ultrasonic field at the end of the plate, as shown in *Figure 2*.

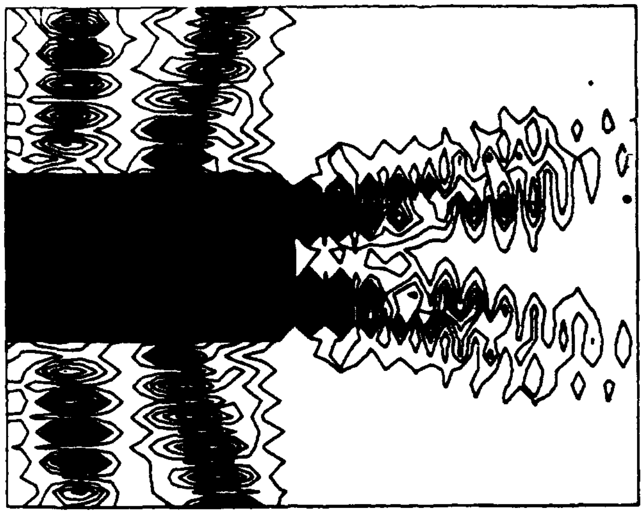

Figure 1 Calculated intensity distribution of an ultrasonic field in water created by reradiation of an A_1-Lamb mode in an aluminium plate. Plate thickness d = 0.9 mm and parameter fd = 2.96 MHz mm

Figure 2 Stroboscopic Schlieren image of radiation lobes for an aluminium plate in water. Parameters are the same as used for Figure 1

This image was obtained by creating a Schlieren image of the ultrasonic field where the light source consisted of a small HeNe laser whose output was modulated by a Bragg cell. The time delay between the generation of the leaky Lamb mode and the stroboscopic illumination of the resulting radiation field could be adjusted. The image presented here shows the radiation field created by an incident 'tone burst' consisting of four cycles, with the

frequency, mode, and plate dimensions being the same as used for the calculations (*Figure 1*). Also visible in *Figure 2* are portions of the transmitted and reflected tone burst, consisting of long, straight wave fronts. Their interference with reradiation lobes, can be held to a minimum when the tone burst is short.

A comparison of *Figures 1* and *2* shows that it is possible to calculate and thus predict radiation patterns and lobe structures of plates immersed in a liquid and vibrating in a given Lamb mode. However, the calculation can also predict the relative phase of the waves in the radiation lobes.

Figure 3 shows the amplitude distribution in the two lobes generated at the narrow end of the plate with only the positive x-direction particle displacements considered. The negative-going half cycles of the waves have been suppressed in the graph and thus it becomes evident that the waves in the two lobes are 180° out of phase. The contour lines within every section of the waves indicate the levels of amplitude in ten equal steps from the maximum to zero amplitude with the lowest contour corresponding to the location of the zero-amplitudes of the waves. The wavefronts show an increasingly noticeable curvature with the circular pattern being centred at the centre of the narrow end of the plate.

Radiation lobes from the top and bottom surfaces of the plate are visible in *Figures 1* and *2*. The points where the two lobes at the top and the bottom of the plate originate are separated by one half-wavelength of the leaky A_1-Lamb mode. Results obtained with the FDM indicate that these lobes have intensity variations along

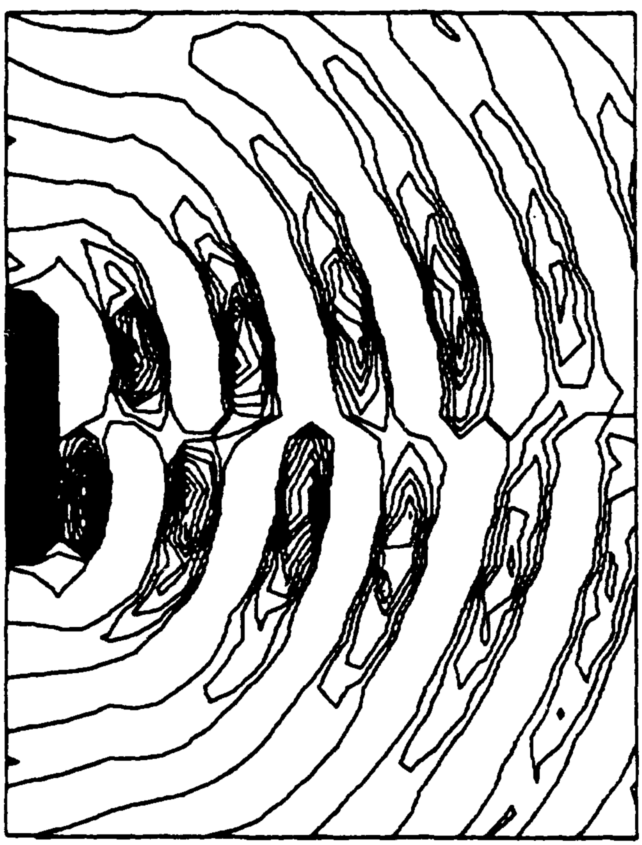

Figure 3 Calculated wavefront pattern for radiation from the narrow plate end considering only positive displacement amplitudes

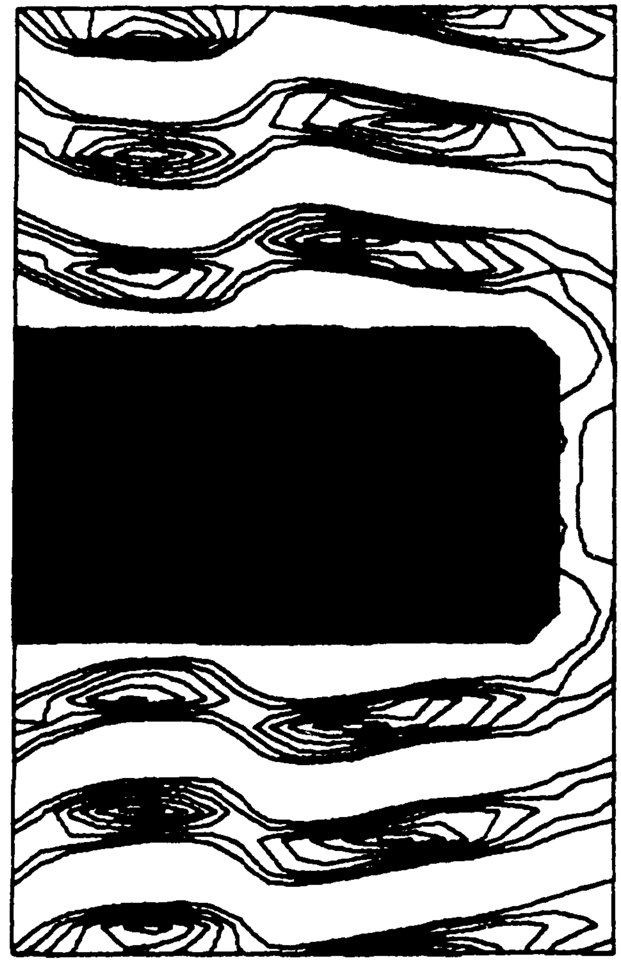

Figure 4 Calculated wavefront pattern for radiation from the top and bottom surfaces, considering only positive displacement amplitudes

their propagation directions which are cyclic. The lobes originating at the plate top and bottom surfaces, as shown in *Figure 1*, have high intensities at the point of origin for the lobes close to the narrow end of the plate while the lobes originating farther away from the end of the plate have a low intensity at the point of origin. However, if the time increment between generation and radiation of the Lamb mode is changed the resulting high intensity areas in the lobes move along the propagation direction.

The phase relationship between the wavefronts within the lobes originating on the top and bottom surfaces as well as the narrow end of the plate can be seen by comparing the intensity distribution in the ultrasonic field (*Figure* 1) with the amplitude distribution (*Figures 3* and *4*) which shows that the waves in adjacent lobes are 180° out of phase. *Figures 3* and *4* reveal the phase information which is not available from an observation of the experimental results shown in *Figure 2*, or from the results of intensity calculations shown in *Figure 1*.

Acknowledgement

One of the authors (WGM) gratefully acknowledges financial support by the Physics Division, Office of Naval Research, US Navy.

References

1 **Zhu, Q.-G., Ruf, H.J. and Mayer, W.G.** *Ultrasonics* (1991) **29** 459
2 **Paraige, P., Luppé, F. and Ripoche, J.** *J Acoust Soc Am* (1991) **89**, pt. 2, 1992
3 **Schoch, A.** *Acustica* (1952) **2**, 18
4 **Neubauer, W.G.** *J Appl Phys* (1973) **44**, 48
5 **Bertoni, H.L. and Tamir, T.** *Appl Phys* (1973) **2**, 157
6 **Pitts, L.E., Plona, T.J. and Mayer, W.G.** *IEEE Trans Son Ultrason* (1977) **24**, 101
7 **Ngoc, T.D.K. and Mayer, W.G.** *J Acoust Soc Am* (1980) **67**, 1149
8 **Bond, L.J.** *Ultrasonics* (1979) **17**, 71
9 **Harker, A.H.** *J NDT* (1984) **4**, 89
10 **Alterman, Z.S. and Loewenthal, D.** *Geophys Roy Astron Soc* (1970) **20**, 101

On the crossing points of Lamb wave velocity dispersion curves

Qigao Zhu and Walter G. Mayer
Ultrasonics Research Laboratory, Physics Department, Georgetown University, Washington, DC 20057

(Received 22 September 1992; accepted for publication 9 December 1992)

Standard dispersion curves relating the phase velocity of Lamb waves to the frequency-plate thickness parameter fd indicate that in some cases symmetrical and antisymmetrical mode velocity curves cross each other. Viktorov's equations are used to show that the crossing points are points where the dispersion curves are discontinuous so that no distinct Lamb mode exists for these particular velocity-fd combinations. Procedures are given to predict how many such points exist and where they are located in a given range of fd.

PACS numbers: 43.35.Pt

INTRODUCTION

Lamb modes are either symmetric or antisymmetric and their velocity of propagation depends on the frequency of the mode f and the thickness d of the solid plate supporting the mode as well as on the bulk wave velocities of the material. If one plots the phase velocity of the various symmetrical Lamb modes as a function of the product $f \cdot d$ one obtains a family of curves none of which crosses another. The same is true for a plot of the antisymmetrical modes. However, if the two families of curves are plotted together there are some instances where symmetric and antisymmetric curves seem to cross, as is evident from a representative example shown in Fig. 1 which applies to aluminum plates.

This implies that the plate supports a symmetrical and an antisymmetrical mode at the same time, both modes propagating with the same phase velocity, if the plate thickness d and the mode frequency f is such that the product fd is the fd value of the "crossing point." The resulting particle displacement within the plate could then be a superposition of two different modes with the vibrational mode of the plate not being defined as a Lamb mode. If one were to assume that no pure Lamb mode exists at crossing points, the defining equations given by Viktorov[1] should not be satisfied for those particular sets of parameters. Viktorov's basic wave equations (I.2) lead to a 4×4 determinant, which in turn lead to two characteristic equations, determining the eigenvalues of the wave numbers of the modes, given by Viktorov as Eqs. (II.4) and (II.5). One can now use these equations to calculate the value of the defining determinant for the values of $f \cdot d$ and the phase velocity of the Lamb modes at the crossing point.

A computation was performed, using double precision, of the value of the determinant for the crossing point located near the value of $f \cdot d = 8.5$ MHz mm and the phase velocity of around 6.9396 km/s, shown in Fig. 1. These calculations for the value of the determinant were performed by changing the values of $f \cdot d$ in increments of 10^{-8} MHz mm and c in increments of 0.1 mm/s. The result is that for the velocity range $c = 6.9396744$ to $c = 6.9396764$ km/s and the $f \cdot d$ range from 8.45435146 to 8.45435157 MHz mm the determinant is nonzero. This result would indicate that a Lamb mode does not exist at points where the velocity dispersion curves for the symmetrical and the antisymmetrical modes cross.

The calculations leading to this result are rather involved and would be very taxing if all the crossing points had to be found with no prior knowledge of the approximate location of these points. Fortunately, a much shorter method is available to determine the location of those points on the families of velocity dispersion curves where no solution to the defining Lamb equations exist. This method will be discussed below.

I. THEORETICAL CONSIDERATIONS

Noting that the phase velocities of Lamb modes at critical crossing points are greater than the bulk longitudinal wave velocity, one can use the defining Lamb mode equations for the symmetrical and antisymmetrical modes, respectively, as given by Viktorov[1]

$$\tan \alpha / \tan \beta + B = 0, \tag{1}$$

$$\tan \beta / \tan \alpha + B = 0, \tag{2}$$

where

$$\alpha = (\pi p f d)/c_t, \tag{3}$$

$$\beta = (\pi q f d)/c_t, \tag{4}$$

$$p = \sqrt{1-(c_t/c)^2}, \tag{5}$$

$$q = \sqrt{(c_t/c_l)^2-(c_t/c)^2}, \tag{6}$$

$$B = \frac{4(c_t/c)^2 pq}{[2(c_t/c)^2-1]} > 0, \tag{7}$$

with c_t the shear wave velocity of the bulk solid, c_l the longitudinal wave velocity, and c the phase velocity of a Lamb mode defined by Eqs. (1) and (2).

As written, Eqs. (1) and (2) apply as long as $c > c_l$. This is the region of interest for the present purpose. One can write general expressions for the defining equations

$$S(c,fd) = \tan \alpha / \tan \beta + B, \tag{8}$$

$$A(c,fd) = \tan \beta / \tan \alpha + B. \tag{9}$$

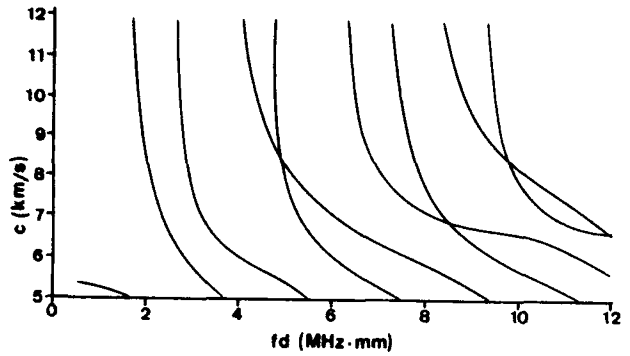

FIG. 1. Section of Lamb mode velocity dispersion curves for aluminum showing the phase velocity c and thickness-frequency parameter fd for the first four mode crossing points.

Solutions of Viktorov's equations are determined by the roots of $S(c,fd)$ and $A(c,fd)$, i.e., the values of c and an appropriate fd are defined for an S mode and an A mode when $S(c,fd)$ or $A(c,fd)$, respectively, are zero.

II. CRITICAL POINTS OF VIKTOROV'S EQUATIONS

The variables α and β are the functions of the parameters c and fd. Thus, for any given phase velocity c, $\tan\alpha$ and $\tan\beta$ are periodic functions of fd, where the periodicities are c_t/p and c_t/q, respectively. The arguments of both tangent functions are zero for $fd=0$, regardless of the value of c. With increasing values of fd, the values of $S(c,fd)$ and $A(c,fd)$ usually remain nonzero; however, in order to satisfy Viktorov's equations they must be zero which occurs only when either $\tan\alpha/\tan\beta=-B$ or $\tan\beta/\tan\alpha=-B$ according to Eqs. (8) and (9).

These conditions on Eqs. (8) and (9) must be satisfied simultaneously at mode crossing points in the c-fd plane if an A mode as well as an S mode are to exist at such points. Investigating the behavior of the two tangent functions one finds that two types of critical points may exist where there are no solutions to Viktorov's equations. As will be shown below, these critical points in the c-fd plane are identical to the mode crossing points; one of these points was identified in the introductory discussion of the determinant solution.

Type I critical points exist when the respective values of $\tan\alpha$ and $\tan\beta$ both reach infinity at the same value of fd which occurs when

$$\alpha=(\pi pfd)/c_t=(m+\tfrac{1}{2})\pi, \qquad (10)$$

$$\beta=(\pi qfd)/c_t=(n+\tfrac{1}{2})\pi, \qquad (11)$$

for certain values of the integers m and n and for certain values of p and q which in turn are functions of the phase velocity.

Type II critical points exist when $\tan\alpha$ and $\tan\beta$ reach zero at the same value of fd that requires that the arguments are

$$\alpha=(\pi pfd)/c_t=m\pi, \qquad (12)$$

$$\beta=(\pi qfd)/c_t=n\pi. \qquad (13)$$

The procedure to determine at what values of c, fd, and m and n type I and type II critical points occur for a set of Lamb mode velocity dispersion curves consists of combining Eqs. (10) and (11) for the type I points, and Eqs. (12) and (13) for type II critical points.

As an example consider the procedure for type I critical point evaluation for an aluminum plate where the longitudinal and shear bulk wave velocities are, respectively, 6.42 and 3.04 km/s. These are the same values that were used in the determinant calculation mentioned in the Introduction above.

Treating m and n as integers whose magnitude is not specifically assigned, Eqs. (10) and (11) are two equations with two unknowns, i.e., a critical value of fd and a critical value of c in p and q, respectively. Not specifying for the moment the value of fd but knowing that fd will have to be the same in α and β where a critical point is located, one can substitute the expression for fd from Eq. (10) into Eq. (11) and solve for c in terms of m and n. This yields the following expression for the critical phase velocity, c_c:

$$c_c=\sqrt{\frac{(m+\frac{1}{2})^2-(n+\frac{1}{2})^2}{(m+\frac{1}{2})^2(c_t/c_l)^2-(n+\frac{1}{2})^2}}\,c_t. \qquad (14)$$

For the critical phase velocity to be real, nonzero, and greater than the longitudinal wave velocity, the conditions for m and n are

$m\neq n,\quad m>n,\quad (m+\frac{1}{2})/(n+\frac{1}{2})>c_l/c_t=2.12$ for Al.

For these conditions and the condition that $\tan\alpha$ and $\tan\beta$ go to infinity at the same fd, one finds only three (m,n) combinations, (1,0), (2,0), and (3,0) for the range of the parameter fd given by $0<fd<12$.

Substituting these three sets of values for (m,n) into Eq. (14) yields three critical phase velocities. The three corresponding critical fd values are found by substituting the three sets of c_c and (m,n) into either Eq. (10) or Eq. (11) to solve for the value of fd. The three critical points for aluminum are thus found at

$c=8.5221$ km/s, $fd=4.88$ MHz mm, $(m,n)=(1,0)$,

$c=6.9396$ km/s, $fd=8.45$ MHz mm, $(m,n)=(2,0)$,

$c=6.6647$ km/s, $fd=11.96$ MHz mm, $(m,n)=3,0)$.

Type II critical points are determined in a similar fashion by using Eqs. (12) and (13). Following essentially the same procedure as outlined above, one finds the possible critical velocities to be given by

$$c_c=\sqrt{\frac{m^2-n^2}{m^2(c_t/c_l)^2-n^2}}\,c_t. \qquad (15)$$

Making the appropriate substitutions one finds only one (m,n) set in the range $0<fd<12$. This single type II critical point has the values

$c=8.5221$ km/s, $fd=9.76$ MHz mm, $(m,n)=(3,1)$.

Comparing these results with the mode velocity curves of Fig. 1 shows that the four calculated critical points are the only crossover points of A and S modes in the range

$0 < fd < 12$. Inasmuch as the parameters of the critical points do not satisfy Viktorov's defining equations, the crossover points are not valid solutions either and neither an antisymmetrical nor a symmetrical mode exists at these points.

III. CONCLUSION

The Lamb mode velocity dispersion curves generally exhibit a number of points where an A mode and an S mode cross which seems to imply that for that particular set of parameters a solid plate should be able to support a symmetrical as well as an antisymmetrical mode. Examining the simplest form of Viktorov's equation shows that the parameters defining these crossing points do not constitute a solution to the equations, thus these unique points are not parts of either the symmetrical or antisymmetrical velocity dispersion curves. The relatively simple procedure described here can be used to find the number of crossing points of Lamb mode velocity dispersion curves as well as their velocity and fd parameters of these points.

ACKNOWLEDGMENT

This work was supported by the Office of Naval Research, U.S. Navy.

[1] I. A. Viktorov, *Rayleigh and Lamb Waves: Physical Theory and Applications* (Plenum, New York, 1967).